AD-A104 878 GEORGIA INST OF TECH ATLANTA SCHOOL OF INDUSTRIAL AN--ETC F/G 12/1
DEVELOPMENT OF A BASIC METHODOLOGY FOR USE IN ANALYZING FEASIBL--ETC(U)
JUN 81 H D RATLIFF
N00014-79-C-0035
UNCLASSIFIED
NL

1 OF 1
AD
A104878

END
DATE FILMED
10-81
DTIC

LEVEL II

12

FINAL REPORT

DEVELOPMENT OF A BASIC METHODOLOGY FOR USE IN ANALYZING FEASIBLE SCHEDULING AND ROUTING SCHEMES

Submitted By

H. Donald Ratliff
Principal Investigator

Submitted To

OFFICE OF NAVAL RESEARCH
Arlington, Virginia 22217

Under

Contract No. N00014-79-C-0035

June 1981

GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL OF INDUSTRIAL & SYSTEMS ENGINEERING
ATLANTA, GEORGIA 30332

FINAL REPORT

CONTRACT NO. N00014-79-C-0035

DEVELOPMENT OF A BASIC METHODOLOGY FOR USE IN ANALYZING FEASIBLE SCHEDULING AND ROUTING SCHEMES

Submitted To

Office of Naval Research
Arlington, Virginia 22217

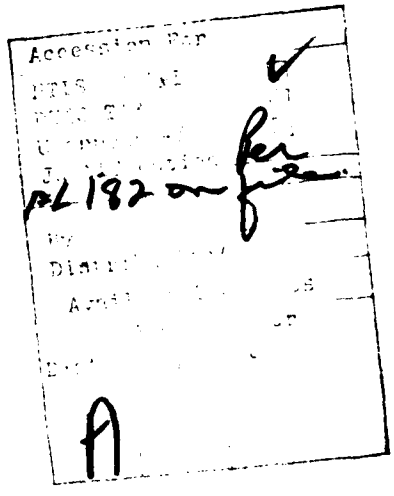
Submitted By

H. Donald Ratliff
Principal Investigator

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

June 1981

TABLE OF CONTENTS



I. INTRODUCTION

The focus of this research has been primarily on the development and testing of interactive methodology for attacking very complex and difficult scheduling and routing problems. The research has involved three major components; (1) determining the level of human interaction, (2) developing an interactive interface, and (3) developing mathematical models to aid in the decision making process. The interactive interface has been built around a colorgraphics computer terminal. Two such units have been purchased by the School of Industrial and Systems Engineering at Georgia Tech as a result of research sponsored under this contract. Developing the interactive interface and determining the level of human interaction have been the most time consuming activities since there has been very little previous research on which to build in this area.

While we have found that to be effective, interactive methodology must be tailored to the particular application, there are some unifying underlying concepts. The first is that some spatial representation of the problem is essential if the human is to play a major role in the optimization phase of the decision making process. If the role of the human only involves selecting from a few alternatives generated by an optimization model or heuristic, numerical information may suffice. However, if the human is to help guide the optimization process, human insight regarding the optimization process is much better when aided by some spatial representation. For scheduling and routing problems, we have found bar charts of resources versus time and network representations to be very insightful. While these representations include numerical information (e.g., time, capacity, distance, ets.) the primary insight seems to come from the picture rather than the

numbers.

In developing our methodology, we have looked at three specific problem areas: (1) long range fleet scheduling, (2) vehicle delivery, and (3) order picking. These topics are discussed in detail in Sections II, III, and IV.

II. FLEET SCHEDULING

The long range fleet scheduling problem is a very important but very difficult problem. It is difficult from a mathematical point of view because it has a very complex constraint structure and a huge number of possible schedules. It is also difficult from a philosophical point of view since the objective is not precisely defined and tends to vary as conditions change. The fleet scheduling problem can best be described in terms of a specific class of ships. We will talk in terms of the cruisers and destroyers which are a part of the Atlantic Fleet.

Very briefly the SURFLANT long-range CRU/DES scheduling problem is as follows. Roughly 70 destroyers and 10 cruisers are to be scheduled over a 60 month planning horizon. There are a number of "hard" commitments (e.g., SIXTHFLT, NATO, etc.) which must be satisfied. There are other commitments (e.g., service to other fleet training) which are "soft" in the sense that they are satisfied when possible but not at the expense of the hard commitments. Each unit has a periodically scheduled overhaul. There are numerous "constraints" on deployments. For example, a unit should not be redeployed for at least six months following a deployment with eight or nine months desirable. Units require a workup period of six or seven months after overhaul before deployment with eight or nine months desirable. Deployment groups should, if possible, have geographical integrity. While the above properties are incomplete, they do indicate the flavor of the scheduling problem.

This problem is currently being solved using pencil and paper methods. There are some obvious drawbacks to using manual methods to attack scheduling problems of this size and complexity. Some of these shortcomings are vividly indicated by LCDR Bobst's comments in Appendix C of [3]. Manual methods

are very time consuming, hence, it is difficult to react to changes in the system. A great deal of insight into the scheduling problem is required. This insight is very difficult to document and in many cases can be obtained only by extensive hands on experience. It is impossible to consider more than a very small fraction of the possible schedules. And perhaps most importantly, in cases where the system constraints are "tight" a scheduler may not be able to generate a feasible schedule even after many hours of effort.

Given these difficulties with manual scheduling methods, it seems natural to consider computerization of at least a portion of the scheduling function. The first level of computerization is obviously some form of "information system." This is in simple terms a bookkeeping system for keeping track of the various parameters relevant to the scheduling problem. One such system, which is apparently operational, is described in DPSCPAC documents [1] and [2]. The report writing capability of this system allows information about the schedule to be printed out in various forms including a sort of modified Gantt chart. The system is not, however, a schedule generator. The schedule must be inputted by the user. The system has the flexibility to allow the user to create schedules from a set of standard modules but the modules themselves must be provided by the user. The system being proposed in ORI report [3] is also an information system rather than a schedule generator.

These systems have some advantages over manual methods. In particular, they allow the user to very rapidly access information about given schedules and they allow this information to be organized in a variety of forms. They force a degree of "standardization" which may be desirable. And, in theory at least, they make it possible for the scheduler to more easily

check schedule feasibility and hence allow the consideration of more potentially good schedules.

They have as potential disadvantages the requirements that the user must be knowledgeable about the computer system in addition to being knowledgeable about the scheduling problem itself. Also, the user must obviously have access to a computer. In order to have any real effectiveness, this access needs to include an interactive capability. To some extent, these difficulties will be encountered in any form of computer aided scheduling.

The greatest shortcoming of information systems of the type indicated above is that they only have the capability to speed up the scheduling process. They do not bring to bear any of the available mathematical power to overcome problems the user must face in resolving the fundamental combinatorial structure inherent in the scheduling problem.

Mathematical Model

In order to determine the respective roles of the computer model and the human in an interactive process, it must first be ascertained how much of the problem structure can be captured in a computationally tractable mathematical model. After examining a number of alternative formulations, it was determined that a generalized set partitioning model was the most appropriate for modeling this problem.

In order to illustrate the generalized set partitioning structure consider the following simple example. (The structure generalize in a straight forward fashion as will be indicated later.) Suppose that there are five destroyers to be scheduled. Destroyers 1 and 2 are based at Charleston while destroyers 3, 4, and 5 are based at Norfolk. Suppose that there is a single commitment to be satisfied. Suppose that the commitment is to provide two destroyers at all times. Let the planning horizon be

twelve months.

We will enumerate the possibilities for each destroyer in a sort of expanded Gantt Chart as illustrated in Figure 1. Suppose that Destroyer 1 can complete all of its desired workup and then deploy during April through September. (The possible deployments are represented in rows 1 through 12 of Figure 1.) This deployment is represented by column x_{11}. Another possibility is to decrease the desired workup by one month and deploy destroyer 1 during March through August. This is represented by colunm x_{12}. A third possiblity is to decrease the desired workup by two months and deploy destroyer 1 during February through July. This is represented by column x_{13}. Constraint 13 insures that we don't choose more than one of the possiblities for destroyer 1.

Suppose that destroyers 3 and 4 are currently deployed. They can continue their deployment through February, have six months off and be redeployed during September through December. These deployments are represented by column x_{31} for destroyer 3 and x_{41} for destroyer 4. Another alternative might be to cut a month off their current deployment (i.e. continue their deployments through January), give them six months off and redeploy them during August through December. These deployments are represented by columns x_{32} and x_{42}. Similarly, deployments for destroyers 2 and 5 are represented by columns x_{21}, x_{22}, x_{51}, x_{52}, and x_{53}.

By utilizing this structure, we can include most of the essential elements of the scheduling problem. Having a longer planning horizon and more than one requirements area simple increases the size of the problem. There are obviously some complex trade-offs which may have to be made in the schedule generation process. For example, it may not be possible to generate a schedule which has both the property that deployment groups have area integrity and each unit has the desired workup period prior to

	x_{11}	x_{12}	x_{13}	x_{21}	x_{22}	x_{31}	x_{32}	x_{41}	x_{42}	x_{51}	x_{52}	x_{53}		
(1)					1	1	1	1	1		1	1	= 2	January
(2)			1	1	1	1		1		1	1	1	= 2	February
(3)		1	1	1	1					1	1	1	= 2	March
(4)	1	1	1	1	1					1	1	1	= 2	April
(5)	1	1	1	1	1					1	1	1	= 2	May
(6)	1	1	1	1	1					1	1		= 2	June
(7)	1	1	1	1						1			= 2	July
(8)	1	1				1		1					= 2	August
(9)	1					1	1	1	1				= 2	September
(10)						1	1	1	1				= 2	October
(11)						1	1	1	1				= 2	November
(12)						1	1	1	1				= 2	December
(13)	1	1	1										= 1	
(14)				1	1								= 1	
(15)						1	1						= 1	
(16)								1	1				= 1	
(17)										1	1	1	= 1	
(18)			1							1			= 1	
(19)				1						1			= 1	

Figure 1: Illustration of the Integer Program Underlying the Scheduling Generator

deployment. To handle such trade-offs in the model, a system is required for weighting the alternatives. These weights can then be used to "drive" the schedule generator.

Since there is a column in the model corresponding to every possible individual ship schedule, there are potentially a very large number of columns. The tractability of the model is primarily a function of the number of columns, hence, it was necessary to determine the approximate number of columns one could expect in an application. Unfortunately, this turned out to be a very difficult task. The only way to accurately make this determination is to actually generate the possible schedules using realistic data.

Some historical data was obtained with the help of the SURFLANT CRU/DES schedulers. Using this data a schedule generator was developed and tested to determine the number of potential possible schedules. The generator included realistic features such as the ability to not allow consecutive deployment to certain area such as the Middle East, minimize and maximum times between deployments, and different overhaul cycles. The sample historical data included 57 possible deployments over a four year planning horizon. The planning horizon was divided into 96 two-week periods.

In general we found that, as expected, the number of possible schedules for each ship varied depending on the minimum and maximum time allowed between deployments and the timing of the overhaul during the period. For the data we examined the ships each had one overhaul during a four year period. If it occurred at the beginning or end of the four year planning horizon then, there were more possible schedules than if it occurred in the middle. Assuming that the time between deployment was between five and ten months, the average number of schedules for the cases considered was 139 with a high of 165 and a low of 92. This means that in

scheduling 80 ships, if all possible schedules for each individual ship were generated, the model would have about 10,000 columns.

Since this is quite large for an integer program, even one with the special set partitioning structure, we felt it necessary to look at heuristic as well as optimum methods for solving the model. These methods are based on a pricing theorem and are essentially the same for both scheduling and routing problems. This will be discussed in detail in Section III.

In general, we found the methods that we developed for solving the set partitioning problem to be very satisfactory. They provided high quality answers in a very reasonable amount of time. Given the imprecise nature of the values or weights assigned to each column, we feel that the effort required to allow the methods to verify optimality is not justified.

Weighting Alternative Schedules

In utilizing the set partitioning model, one of the essential elements is a scheme for determining the value or "weight" which should be given to each column. In fact, any scheme which could be done without human interaction would require such a weighting scheme. The SURFLANT schedulers who have had one original contact at Norfolk felt that such a scheme could be developed. However, as we attempted to develop such a scheme and discussed the concept with CMDR Lee Pittman, who had recently taken over the CRU/DES scheduling activity at SURFLANT, it became clear that a general scheme was not possible. This was due to both the continuously changing nature of the problem and the level of judgement required to evaluate each alternative schedule. We determined that the only realistic way to evaluate an individual ship schedule was to have an experienced

scheduler actually look at it. To generate all of the 10,000 possible individual schedules and have a person evaluate each one was obviously not practical. This lead us to look at interactive schemes where a person would help in the generation of individual schedules (i.e., columns in the model). By performing this column generation interactively, something is lost in terms of optimality since only a small subset of the possible columns would ever be generated. However, the person will hopefully be able to generate very good candidate columns which will then result in a good overall schedule.

Since the underlying problem structure and the set partitioning model are essentially the same for both the fleet scheduling problem and a broad class of delivery problems, it was decided to address both problem classes in developing the interactive methodology. This had two distinct advantages. There were results available in the literature on the delivery problem against which an interactive approach could be compared while none were available for the fleet scheduling problem. Also, the delivery problem is generally simpler than the fleet scheduling problem. We felt that some experience was needed in developing interactive methodology for the simpler problem before attacking the more complex fleet scheduling problem.

The methodology developed for the delivery problem is discussed in Section III. We feel that this represents a real breakthrough in attacking this class of problems. A further refinement of this methodology is being continued under a current ONR contract and will be reported on in the near future. We felt that this same basic methodology with a different interactive interface could be extended to solve the fleet scheduling problem.

Fleet Scheduling Interactive Interface

The first interface that we considered was a bar chart color coded to indicate the various deployment areas. Each ship is represented as a bar with time periods (two weeks) along the horizontal axis. One example of the colorgraphics display is shown in Figure 2. This system allows the user to add or delete deployments to a given ship's schedule, to move deployments from one ship to another, to specify overhaul cycles, and to generate various statistics such as minimum or average time between deployment for a given ship. This is very similar to the manual bar charts currently used to generate schedules. While certain features such as the ability to generate statistics were very helpful, overall the display did not convey enough information to allow the human to bring his insight to bear on the problem. In particular it was designed to handle individual ships while the scheduler is most frequently concerned with developing battle groups.

Hence a second and much more elaborate interface was designed. We proposed that the entire scheduling function be translated into a minicomputer-based interactive scheduling program package and that this package be designed to use the extensive colorgraphics capabilities of the Chromatics CG 1999 minicomputer. The package would consist of four major programs and a master module with menu selection from the master to the four major proqrams. The modules are described in Figure 3.

Ship File Manager. This module is used to maintain a file on diskette of all hulls in the Fleetpak data base. The program functions supported are List, Display, Add, Delete, Change, and Return. Table 1 shows the list of data elements that will be maintained for each ship. Note that the initial version of Fleetpak contains only the first seven data elements.

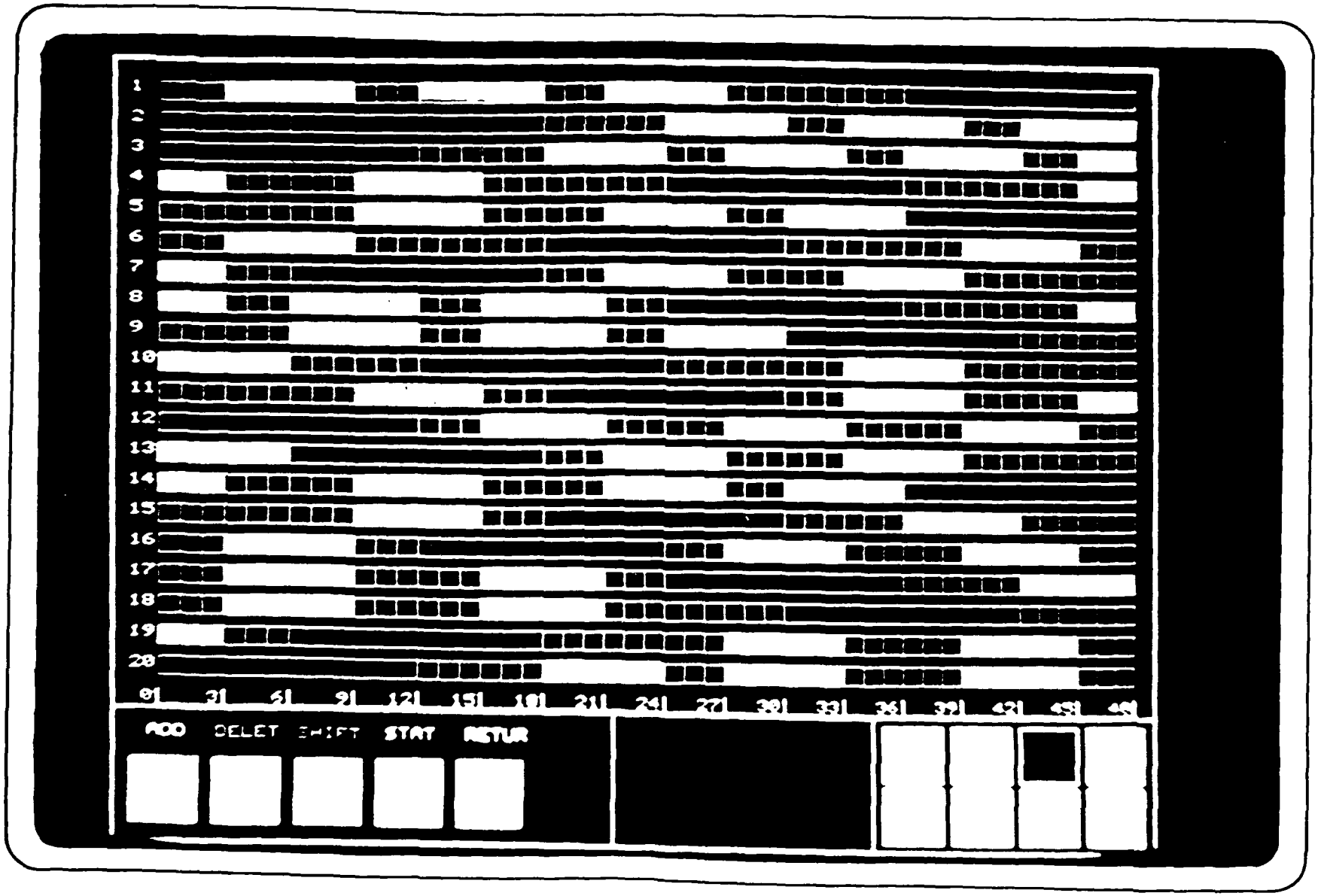


Figure 2. Colorgraphics Display by Ship

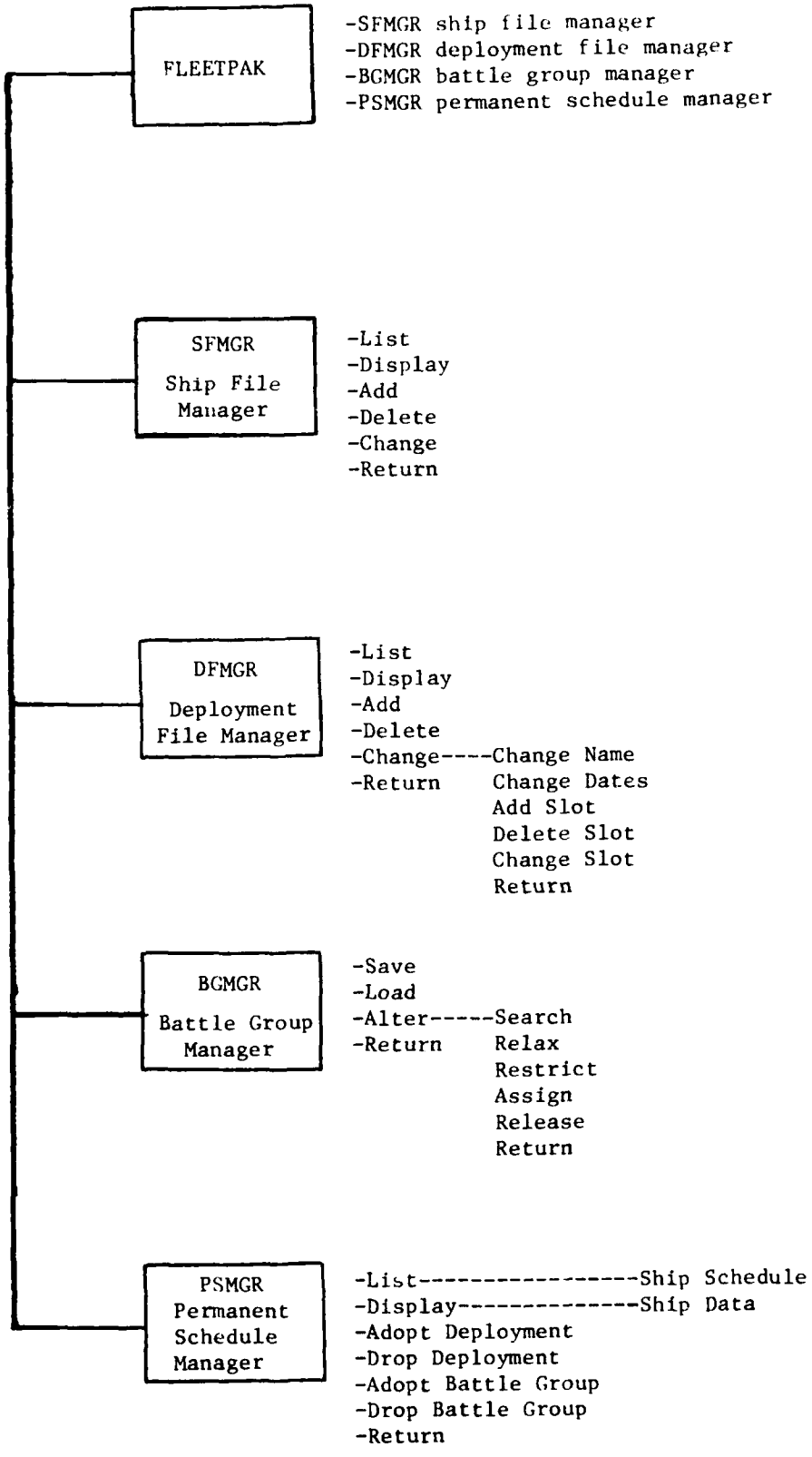


Figure 3. Fleetpak Structure

Table 1. Ship File Contents

Data Element	Bytes Required	
Ship Tupe	4	these elements in initial version of Fleetpak
Hull Number	4	
Ship Name	25	
Ship Class	25	
Characteristic Word	8	
UIC		
Home Port CO		
Home Port		
CO		
ADSCN		
Displacement		
Length		
Beam		
Draft		
Officers		
Enlisted		
Sq Ft Capacity		
Cu Ft Capacity		
Max Speed		
Econ Speed		
Max Speed Range		
Econ Speed Range		
Propulsion		
Number of Shafts		
Fuel Type 1	Fuel Type 1 Capacity	
Fuel Type 2	Fuel Type 2 Capacity	
Fuel Type 3	Fuel Type 3 Capacity	
Fuel Type 4	Fuel Type 4 Capacity	
Number on Board	for all systems	

Later versions may support the entire list.

The Fleetpak user would necessarily use the Ship File Manager as the first step in building a schedule. All the information about available ships would be entered. Later, changes, additions, and deletions would be processed. At any time, information about an individual ship can be displayed or listed. A list of all ships would also be provided.

The logical structure of the ship file is a stack. The physical organization is sequential. Access methods supported are sequential and hashed lookup with sequential lookup of overflows in an overflow table at the end of the file.

Deployment File Manager. This module is used to maintain a file on diskette of all deployments in the data base. The program supports List, Display, Add, Delete, Change, and Return functions. Changes possible include Change Name, Change Dates, Add Slot, Delete Slot, Change Slot, and return. The Change Slot function includes Change Type, Change, Class, Add Characteristic, and Delete Characteristic.

The deployment file manager program is used to define all the available ship activities to which ships could be assigned singly or in battle groups.

Table 2 shows the contents of the Deployment File. Table 3 show the related file called the Event File. No direct user access to the Event File is possible. It is an index file used by the various modules of Fleetpak. The Deployment File provides access to schedule data by deployment name. The Event File provides access by ship type: hull number. Two files are needed to give quick access. For example, it would be very slow to try to find all the deployments scheduled for an individual ship by reading through all deployments and looking within them for an entry for the individual ship.

Table 2. Deployment File Contents

Data Element	Bytes Required	
Deployment Name	8	
Description	40	
Start Year	2	
Start Month	2	
Start Day	2	
End Year	2	
End Month	2	
End Day	2	
Long Range/Short Range Code	6	
LR/SR	1	
Number of Slots	2	
Ship Type	4	1 entry for each slot
Ship Class	4	
Required Characteristics Word	8	

Table 3. Event File Contents

Data Element	Bytes Required	
Ship Type	4	
Hull Number	4	
Number of Deployments	2	
Deployment Name	8	
Required Characteristics Word	8	
Start Year	2	1 entry for each deployment
Start Month	2	
Start Day	2	
End Year	2	
End Month	2	
End Day	2	

Battle Group Manager. The Battle Group Manager module is the most important module. The purpose of BGMGR is to give interactive assignment of available ships to open slots in scheduled deployments. This is the third step in building a schedule. An empty deployment, essentially a set of slots, is called up and reviewed. For each slot, ships are selected that match the characteristics selected for that slot. In addition, the ships must be available during the period of the deployment. To assist the operator, BGMGR provides the capability of displaying any individual ship schedule, any other deployment or battle group, or the overall schedule of all deployments. The ship file entry for an individual ship can also be recalled and displayed.

The purpose of having so many different displays is to give the user the opportunity of knowing why ships have already been given other assignments and of making intelligent choices for including ships in a current battle group. Once a slot is being filled, the operator selects the Search function. All ships that match the characteristics word are displayed in column 2 of the Battle Group Manager. From this list, the operator makes a selection. If he is unsure, then he can display the schedule for the ship in question or look at its activity in other deployments.

Within this procedure, the user can either relax or restrict the characteristics required for each slot. Once a satisfactory match is made, an assignment is made. Changes can be made by releasing the ship already assigned to a slot.

Thus, the functions available in the BGMGR module are Save, Load, Alter, and Return. Under the Alter function are subfunctions of Search, Relax, Restrict, Assign, Release, and Return.

Note that the initial version of Fleetpak supports a one to one mapping of battle groups to deployments. Later versions may allow alternate schedules where deployments and battle groups meet on a many to many basis. In any event, a complete schedule consists of a number of adopted deployments and an equal number of adopted battle groups.

It is within the BGMGR Search and Assign functions that the real scheduling is actually performed under Fleetpak. All other functions are essentially just support and report writing functions. It is at this point exactly that optimization can be introduced.

Permanent Schedule Manager. The PSMGR module is used to build and report schedules. Functions supported are List, Display, Adopt Deployment, Drop Deployment, Adopt Battle Group, Drop Battle Group, and Return. Under the List and Display functions there are seven subfunctions. There include Ship Schedule, Ship Data, Deployment Schedule, Deployment Data, Battle Group Data, Permanent Schedule, and Return.

File structure for the permanent schedule file is fixed record length sequential access. The file is very small.

The main colorgraphics display for this system is illustrated in Figure 4. From this display the user has the ability to display and alter characteristics required for each ship in a battle group, display future requirements for ships with these same characteristics, generate candidate ships for each battle group, display the past and future commitments for candidate ships for each battle group, generate the battle groups, generate various statistics related to the battle groups and to individual ships, and change individual deployments as the requirement or ship availabilities change.

The modules were developed for testing on the Chromatics CG 1999 color microcomputer. Our discussion with CMDR Lee Pittman, the SURFLANT scheduler

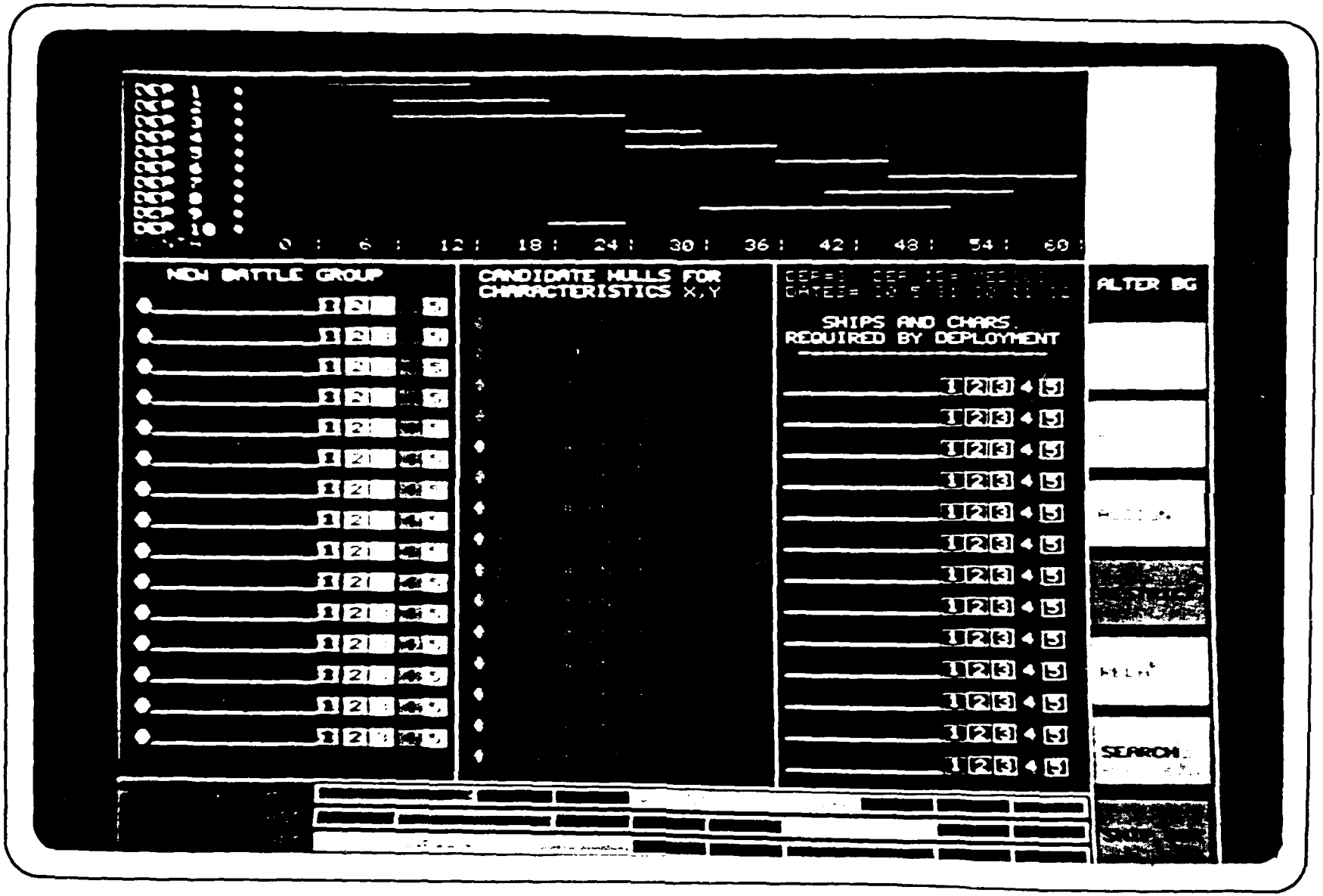


Figure 4. Colorgraphics Display by Battle Group

for cruisers and destroyers lead us to conclude that this system contained the essential elements for efficient fleet scheduling. However, our initial tests indicate that the volume of ship related data required, exceeds the capacity of the microcomputer to process. The design is being modified so that the major data manipulation function will be performed on a host computer. The fundamental interactive methodology which we have developed seems adequate to address this problem. Further refinements are being made under the current ONR contract.

References

1. DPSCPAC System 81, "Long Range Employment Schedule System Command Manual."

2. DPSCPAC System 83, "Long Range Employment Schedule User's Manual."

3. Operations Research Draft Final Report, "Automated Ship Operational Scheduling (AOS) ---Concept Definition Phase," 10 August, 1977.

III. INTERACTIVE DELIVERY

We will consider here a set partitioning based approach for solving a broad class of routing problems. The approach is designed to take advantage of a high level of human interaction; the current implementation is interactive via a colorgraphics display. However, many of the concepts discussed here could be easily implemented in an automatic system.

The routing problem which motivated much of this work is what is called the static pick up and delivery problem. This problem will be discussed in detail in later sections. It is one of the more complex members of the class of routing problems which are amenable to the approach presented here. This class also includes many practical delivery problems. In order to introduce the underlying methodology which provides the basis for the approach, consider a very simple delivery example. Assume that a depot is located at the square box labeled D in Figure 1. From this depot a single delivery is to be made to each of the points represented by numbered circles. The numbers on arcs connecting the circles represents the travel distance between delivery points. Assume also that each vehicle (e.g. truck) can deliver to a maximum of two points on a single trip. The objective is to determine which vehicle should deliver to each point and the routing for the vehicles which minimizes the total distance travelled.

Each column in the matrix of Table 1 represents one possible vehicle route. For example, column one represents a vehicle travelling from the depot to delivery point (1) and returning. The c_j row indicates the

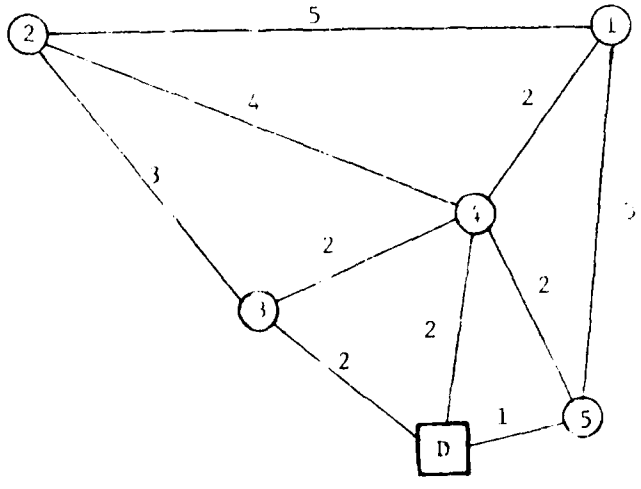


Figure 1. Delivery Example Network

Table 1. Matrix Corresponding to Routes in the Delivery Example of Figure 1

route	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
c_j	8	10	4	4	2	14	10	8	8	10	11	12	6	6	5
	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	0	1	0	0	0	1	0	0	0	1	1	1	0	0	0
	0	0	1	0	0	0	1	0	0	1	0	0	1	1	0
	0	0	0	1	0	0	0	1	0	0	1	0	1	0	1
	0	0	0	0	1	0	0	0	1	0	0	1	0	1	1

distance traveled for each trip. For example, column six represents a vehicle proceeding from the depot to delivery point (1), on to delivery point (2) and from there, back to the depot. The value of c_6 is 14, the total distance traveled for this trip. By enumerating each of the possibilities, as has been done for this matrix, the problem becomes one of selecting a set of columns such that every row is represented in exactly one column and the sum of the costs of the columns selected is the smallest possible. This integer program is called a "set partitioning model".

The set partitioning model was originally proposed for the delivery problem by Balinski and Quandt [2]. The model is very powerful in the sense that many realistic route constraints and cost functions can be handled easily in the column enumeration process. The obvious shortcoming of the model is that there are typically a very large number of columns to be enumerated and the resulting integer program is very large. The approach presented here is heuristic in the sense that we generate only a subset of the possible columns or routes and in general we do not solve the set partitioning model to optimality.

The set partitioning model has two very desirable features for interactive optimization. The first is that any route generated can be included as a column in the model. This allows the human interactor to utilize his/her intuition and spatial preception as well as a wide spectrum of mathematical techniques to generate new routes. The second feature is that, unlike more general integer programs, a feasible solution to the set partitioning model provides the basis for pricing

information which can be used to generate new candidate columns.

We will restrict the class of routing problems considered here to be those for which any subroute of a feasible route is also a feasible route with cost less than or equal to the cost of the original route. By imposing this restriction, as long as there is at least one 1 in every row of the partial set partitioning model that we have enumerated, we can easily generate feasible partitions. The only other restriction that we put on the class of routing problems is that we be able to pose them in a natural way as set partitioning problems. However, it should be noted that if there is not a nice spatial representation of the routing problem, the human interactor is much more restricted in his/her contribution.

Set Partitioning and Row Prices

The set partitioning problem can be stated as

$$\text{minimize} \quad Z = \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j = 1 \quad \text{for } i = 1,2,\ldots,m$$

$$x_j = 0 \text{ or } 1 \quad \text{for } j = 1,2,\ldots,n.$$

For the delivery example in Table 1, the set partitioning model has the second row as the value of the c_j and rows three through seven as the values of the a_{ij}. The variables which are set to one in a solution to the set partitioning problem will be called a "partition." We will denote a partition as $J^k = \{j | x_j^k = 1\}$.

Balas and Padberg [1] provide a recent survey of results related to set partitioning problems.

A fundamental idea underlying much of the work presented here is the concept of "row prices."

Definition: $P^1 = (p_1^1, p_2^1, \ldots, p_m^1)$ is a set of feasible row prices corresponding to the partition J^1 if

$$\sum_{i=1}^{m} p_i^1 a_{ij} = c_j \quad \text{for } j \in J^1.$$

It will be useful to interpret the price p_i^1 as an estimate of the cost to satisfy constraint i using solution X^1. For the delivery problem, p_i^1 is then an estimate of the cost of satisfying the requirement of delivery point i using the route corresponding to partition J^1.

Theorem 1: Given a set of feasible row prices $(p_1^1, p_2^1, \ldots, p_m^1)$ corresponding to partition J^1 with value Z^1, any other partition J^2 has value

$$Z^2 = Z^1 - \sum_{j \in J^2} \sum_{i=1}^{m} (p_i^1 a_{ij} - c_j)$$

Proof:

$$\sum_{j \in J^2} \left(\sum_{i=1}^{m} p_i^1 a_{ij} - c_j \right) = \sum_{i=1}^{m} p_i^1 \sum_{j \in J^2} a_{ij} - Z^2$$

Since J^2 is a partition, $\sum_{j \in J^2} a_{ij} = 1$ for each $i = 1,2,\ldots,m$. Also, since $(p_1^1, p_2^1, \ldots, p_m^1)$ are feasible row prices corresponding to X^1 we have $\sum_{i=1}^{m} p_i^1 = Z^1$. Hence the result follows.

Corrollary 1: For any set of feasible row prices P^1 corresponding to a partition J^1 if $\sum_{i=1}^{m} (p_i^1 a_{ij} - c_j) \leq 0$ for $j = 1,2,\ldots,n$ then X^1 is optimum.

It can also be shown, using linear programming duality, that a set of feasible row prices P^1 satisfying Corrollary 1 exists if and only if X^1 is an optimum solution with the constraints $x_j = 0$ or 1 replaced by $x_j \geq 0$ for $j = 1,2,\ldots,n$.

The quantity $\sum_{i=1}^{m} p_i^1 a_{ij} - c_j$ will be interpreted as the "potential" savings over the value of Z^1 which can result from constructing a partition that includes column j. Note from Theorem 1 that the potential savings can actually be achieved only if a partition can be constructed from columns with nonnegative potential savings.

Potential Savings Heuristic

Given a partition J^1 a corresponding set of feasible row prices P^1, an attractive heuristic for attempting to generate a better partition is the following:

Step (0): Let $J^2 = \emptyset$ (J^2 will be the indices of columns in the new partition) and $N = \{1,2,\ldots,n\}$, (N will be the indices of columns which are candidate for inclusion in J^2)

Step (1): Calculate the potential savings $\sum_{i=1}^{m} p^1 a_{ij} - c_j$ for $j = 1,2,\ldots,n$.

Step (2): Determine $k \in N$ such that $\sum_{i=1}^{m} p_i^1 a_{ik} - c_k \geq \sum_{i=1}^{m} p_i^1 a_{ij} - c_j$ For all $j \in N$ (i.e., pick the column in N with the largest potential savings)

Step (3): If $\sum_{j \in J^2} a_{ij} = 1$ then set $a_{ik} = 0$. (Note from the assumption of section 1 that any subroute of a feasible route is also a feasible route the new column is legitimate)

Step (4): Let $J^2 = J^2 \cup \{k\}$ (i.e., put column k in the new partition)

Step (5): Delete from N all j for which $a_{ik} = 1$ and $a_{ij} = 1$ for some $i = 1,2,\ldots,m$.

Step (6): If $N = \emptyset$ stop. Otherwise go to step (2).

Note that under the assumption that any subset of a route is also a feasible route (discussed in section 1) this procedure will always terminate with J^2 as a partition although not necessarily a better partition than J^1. In addition, computation to date indicates that an optimum or near optimum solution to the set partitioning problem is determined very quickly by repeated application of the potential savings heuristic. The heuristic is repeated until either optimality is proven (i.e., for some X^k all $\sum_{i=1}^{m} p_i a_{ij} - c_j \leq 0$) or until some specified number of partitions has been generated.

To illustrate the potential savings hueristic, consider again the delivery example depicted in Figure 1 and Table 1. Suppose that we select $J^1 = \{1,2,\ldots,5\}$ as an initial partition. A set of feasible row prices P^1 is given in Table 2. (The question of how to generate "good" feasible row prices will be addressed in the next section.) The corresponding potential savings $\sum_{i=1}^{m} p_i^1 a_{ij} - c_j$ are also given in Table 2. Applying the potential savings heuristic and breaking ties by selecting the column with the lowest index yields the new partition $J^2 = \{6,13,5\}$. The new partition has a cost of $Z^2 = 22$ as compared to a cost $Z^1 = 28$ for the initial solution.

Using the row prices P^2 and potential savings $\sum_{i=1}^{m} p_i^2 a_{ij} - c_j$ of Table 2 and reapplying the potential savings heuristic yields the partition $J^3 = \{8,10,5\}$ which has a cost of $Z^3 = 20$. Again, from Table 2 we find that using the row prices P^3 given potential savings $\sum_{i=1}^{m} p_i^3 a_{ij} - c_j \leq 0$ for $j = 1,2,\ldots,n$. Hence, from Corrollary 1 the partition J^3 is optimum.

Row Pricing

For a given partition X^k a set of feasible row prices is obtained by allocating the column cost c_j for each $j \epsilon J^k$ among the rows having $a_{ij} = 1$. For the delivery example, this corresponds to allocating the trip cost among the delivery points of the trip. When a column $j \epsilon J^k$ contains only one $a_{ij} = 1$, the row price is $p_i^k = c_j$. However, when a column j contains more than one $a_{ij} = 1$, there are an infinite number of possible sets of prices. As an example consider the partition $J^3 = \{8,10,5\}$ for the problem in Table 2. Since column 5 has only $a_{5,5} = 1$ the value $p_5^3 = 2$ is unique. Column 8 has both $a_{1,8} = 1$ and $a_{4,8} = 1$, hence the cost $c_8 = 8$ could be allocated between rows 1 and 4 in an infinite number of ways. Similiarly, column 10 has both $a_{2,10} = 1$ and $a_{3,10} = 1$, hence $c_{10} = 10$ could be allocated between rows 2 and 3 in an infinite number of ways. If we allocate c_8 as $p_1^3 = 4$ and $p_4^3 = 4$ and allocate c_{10} as $p_2^3 = 5$ and $p_3^3 = 5$, the resulting $\sum_{i=1}^{m} p_i^3 a_{ij} - c_j$ do not indicate that J^3 is an optimum partition. Hence the set of prices P^3 given in Table 2 are clearly better since they do indicate that J^3 is on optimum partition.

Ideally, we would like a set of prices which would drive the potential savings heuristic toward an improving solution and would

Table 2. An Example Illustrating the Potential Saving Heuristic with $J^1 = \{1,2,3,4,5,\}$, $J^2 = \{6,13,5\}$, and $J^3 = \{8,10,5\}$

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	P^1	P^2	P^3
c_j	8	10	4	4	2	14	10	8	8	10	11	12	6	6	5			
	1					1	1	1	1							8.0	6.2	5.3
		1				1				1	1	1				10.0	7.8	7.1
			1				1			1			1	1		4.0	3.0	2.9
				1				1			1		1		1	4.0	3.0	2.7
					1				1			1		1	1	2.0	2.0	2.0
$\sum_{i=1}^{m} p_i^1 a_{ij} - c_j$	0	0	0	0	0	4	2	4	2	4	1	0	2	0	1			
$\sum_{i=1}^{m} p_i^2 a_{ij} - c_j$	-1.8	-2.2	-1	-1	0	0	-.8	1.2	.2	.8	-.2	-2.2	0	-1	0			
$\sum_{i=1}^{m} p_i^3 a_{ij} - c_j$	-2.7	-2.9	-1.1	-1.3	0	-1.6	-1.8	0	-.7	0	-1.2	-2.9	-.4	-1.1	-.3			

indicate optimality when no improving solution is possible (i.e., We would like the prices to be analogous to dual variables in linear programming). Unfortunately, it is easy to construct cases for which no such prices exist (i.e., any problem for which the integer and continuous solution differ). For the delivery problem and the more complex dial-a-ride problem (to be discussed later), allocating column cost in proportion to the cost of serving the delivery points one-at-a-time is intuitively appealing and seems to work very well. As an illustration consider again the partition J^3 in Table 2. Column 8 has $a_{1,8} = 1$ and $a_{4,8} = 1$. The cost of serving delivery point 1 if it is the only point in a trip is $c_1 = 1$. The cost of serving delivery point 4 if it is the only point in a trip is $c_4 = 4$. The prices for rows 1 and 4 were determined as $p_1^3 = \frac{c_1 c_8}{c_1 + c_4} = 5.3$ and $p_4^3 = \frac{c_4 c_8}{c_1 + c_4} = 2.7$. The other prices in Table 2 were determined similiarly.

Column Generation

For large scheduling and routing problems it is generally not practical to generate all columns of the corresponding set partitioning model. The remainder of this paper will be concerned with using information gleaned from one solution via Theorem 1 to generate a new and hopefully better solution. This is accomplished by either generating new columns, adding them to the current set partitioning model, and then resolving the model or by using the information from Theorem 1 directly to generate a new solution. In the latter case it is not necessary to retain the columns of the set partitioning model. However, if the columns are retained, it is possible to further improve the solution by periodically solving the set partitioning model.

We should note that for the class of scheduling and routing problems being considered here, it is very easy to generate an initial solution. For our examples we use the identity solution (e.g., in the delivery problem this is the solution which has each vehicle making a single delivery) as our initial solution. However, any feasible solution could be used as the initial solution.

Clearly there is a broad spectrum of possible approaches that one might use to generate new columns and/or new solutions to the set partitioning model. In fact, variations of many of the heuristics which have been applied to delivery problems can be used very effectively in conjunction with Theorem 1. In the next section we will discuss the use of the Clarke and Wright [3] savings procedure in conjunction with the delivery problem. In later sections we will discuss more complex clustering and chaining heuristics as we have applied them to the dial-a-ride problem.

Clarke and Wright Procedure with Pricing

The "savings" heuristic of Clarke and Wright [3] is the most widely known of the heuristics developed to date for delivery problems. The algorithm proceeds by calculating a savings for each pair of delivery points i and j defined as

$$s_0(i,j) = 2d_{0i} + 2d_{j0} - (d_{0i}+d_{ij}+d_{j0}) = d_{0i} + d_{j0} - d_{ij}$$

which is the savings in mileage of supplying delivery points i and j on the same route as opposed to supplying them individually directly from the depot (d_{0i} is the distance from the depot to delivery point i).

Routes are then constructed either one-at-a-time or in parallel by considering pairs of points in order of decreasing savings and including them in the same route if such a route is feasible.

Suppose that we consider once more the delivery example in Figure 1 and the covering solution information in Table 2. Note that the Clarke and Wright savings values are exactly the values of $\sum_{i=1}^{m} p_i^1 a_{ij} - c_j$ in Table 2 since $d_{0i} = p_i^1/2$ for $i = 1,2,\ldots,m$. Applying the C-W savings algorithm yields the same routing configuration as $J^2 = \{6,13,5\}$. At this point, the C-W algorithm would terminate. However, suppose that we set $d_{0i} = p_i^2/2$ for $i = 1,2,\ldots,m$. The new savings are then exactly the $\sum_{i=1}^{m} p_i^2 a_{ij} - c_j$ in Table 2. Applying C-W algorithm yields the same routing configuration as $J^3 = \{8,10,5\}$ which as noted previously is the optimum solution to this delivery example. Hence, for this example at least the C-W algorithm without pricing did not yield an optimum solution while the same algorithm when combined with pricing did yield the optimum solution.

Note that when routes are allowed to contain more than two trips, only a small subset of the set partitioning columns would be generated using this procedure. Also, note that the prices and savings can be *calculated without ever generating* the set partitioning matrix.

When the number of points allowed in a route exceeds two, some interesting questions arise as to exactly how the algorithm should be implemented. As an illustration, suppose that in the example we allow a vehicle to deliver to at most three points rather than two. Now suppose that we start with the solution J^3 in Table 2. We see that the two-at-a-time potential savings are all non-positive. However, if we ignore this

fact and proceed with the algorithm, we would put points 1 and 4 in the same route since their potential savings of zero is maximum. If we then consider adding a third point to this route, we find that adding point 5 has a potential savings of 2 which is maximum. Finally, we combine points 2 and 3 into a single route since this has a potential savings of zero. The resulting routes (4,15) and (2,3) has a length of 10. Therefore, it appears that we can get better information by recalculating the potential savings after each augmentation of a route. This recalculation is not done in most implementations of the C-W algorithm.

Clearly, when constructing a route containing more than two points one must decide where in the route to put each additional point. The potential savings can be determined exactly only by solving a travelling salesmen problem over each new point which is a candidate to be added to the route. This is computationally expensive if a route can contain a large number of points. In most implementations of the C-W algorithm, new points are simply added on to the end of the route being constructed. When the algorithm is implemented interactively using computer graphics, it appears that the human can perform an important role both in selecting candidate points and in inserting them logically into routes.

Location - Allocation with Pricing

Another procedure which Krolak and Nelson [5] have found effective in approaching the delivery problem utilizes the location - allocation model of Cooper [4]. This basic concept can also be used in conjunction with Theorem 1 to generate intuitively appealing columns to add to the set partitioning model.

To illustrate the procedure consider the delivery example illustrated in Figure 2. Again the circles represent delivery points and the square represents the depot. The basic idea is to use a surrogate distance

rather than the actual distance in determining the points which are assigned to each vehicle. The surrogate distance is obtained by assuming that the vehicle travels from the depot to a specified cluster point, represented in Figure 2 by the dashed circles. It then makes the deliveries, returning after each delivery to the cluster point. After all deliveries have been made, the vehicle returns to the depot. Under this surrogate distance, the problem becomes one of locating the cluster points, one for each vehicle, and then assigning the delivery points to each vehicle.

If (a_0,b_0) represents the coordinates of the depot, (a_i,b_i) represents the coordinates of delivery point i, (x_j,y_j) represents the coordinates of cluster point j, and assuming Euclidean distance, the problem can be modeled as follows:

$$\min_{x,y,z} \sum_{i=0}^{m} \sum_{j=1}^{n} 2[(x_j-a_i)^2 + (y_j-b_i)^2]^{\frac{1}{2}} z_{ij}$$

$$\text{s.t.} \quad \sum_{i=1}^{m} z_{ij} \leq K \qquad j = 1,2,\ldots,m$$

$$\sum_{j=1}^{n} z_{ij} = 1 \qquad i = 1,2,\ldots,m$$

$$z_{ij} = 0 \text{ or } 1 \quad \text{all } i,j$$

where n is the number of vehicles and K is the vehicle capacity. When $z_{ij} = 1$, delivery point i is assigned to vehicle j and when $z_{ij} = 0$, delivery point i is not assigned to vehicle j.

While there is no method for efficiently solving this problem optimally, the following is an attractive heuristic. Pick a set of

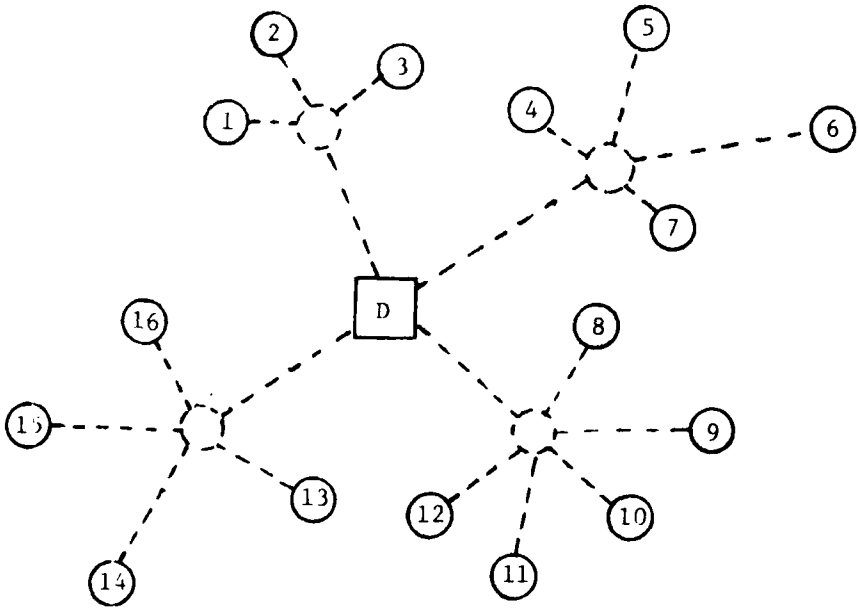


Figure 2. Delivery Example to Illustrate the Location - Allocation Model for Clustering.

locations for the cluster points. With these coordinates fixed, solve the resulting assignment problem. With these values of z_{ij} fixed, solve the resulting location problem. Continue alternating between the assignment and location problems for some specified number of iterations or until there is no further improvement in the objective. Once a cluster has been determined, the vehicle is then routed among the points of the cluster.

A slight modification of this model, together with Theorem 1, allows us to generate attractive new columns for the set partitioning problem. Suppose that we have a solution to the set partitioning problem and a set of row prices $p_1, p_2, \ldots, p_m$. Now consider the model

$$\min_{x,y,z} \sum_{i=0}^{m} \sum_{j=1}^{n} 2[(x_j - a_i)^2 + (y_j - b_i)^2]^{1/2} z_{ij} - \sum_{i=1}^{m} \sum_{j=1}^{n} p_i z_{ij}$$

$$\text{s.t.} \sum_{i=1}^{m} z_{ij} \leq K \quad j = 1,2,\ldots,n$$

$$\sum_{j=1}^{n} z_{ij} \leq 1 \quad i = 1,2,\ldots,m$$

$$z_{ij} = 0 \text{ or } 1.$$

Any cluster generated by this model will have a positive potential savings, with respect to the surrogate distance. Hence, it corresponds to an attractive column to add to the set partitioning problem (considering surrogate distances).

Since the second constraint has been changed to an inquality, not all delivery points will be assigned to cluster points. This simply

means that the current row price for the delivery point is more attractive than the cost of serving the delivery point in alternative clusters considered by the model. The model can be solved using the same heuristic discussed for the earlier location - allocation model.

Dial-A-Ride Problem

The dial-a-ride problem is a specialization of the general pick-up and delivery problem. It is a much more complex routing problem than the simple delivery problem. In the dial-a-ride problem we are given an origin-destination trip matrix and an underlying network on which the trips are to be made. There is a single item (people, goods, etc.) (demand for service) at each origin that needs to be transported to its specified destination. The items are transported from origins to destinations on vehicles each having capacity K. We wish to satisfy the trip requirements while travelling the minimum distance.

This is a "static" version of the dial-a-ride problem since time is not considered. There are a number of more complex versions of this problem, but this version is sufficient to demonstrate the basic ideas of our approach.

The set partitioning model for the dial-a-ride problem is analogus to that of the delivery problem, but here rows represent trips rather than simply delivery points. The vehicle capacity constraints are handled by generating only routes which satisfy them.

Decomposition

In delivery problems, representative of the "one-ended" class of routing and scheduling problems, we need only be concerned with a single stop for a vehicle to satisfy a particular demand for service. In contrast, the "two-ended" class, which includes dial-a-ride problems, requires

two stops in a specific order (sequence) to satisfy a specific demand for service. It is this requirement for sequencing of "pickup" and "dropoff" point pairs which adds greatly to the difficulty of handling the two-ended class of vehicle routing problems.

When one considers examples of two-ended problems, it becomes immediately apparent that the sequencing requirements greatly inhibit the complex pattern processing abilities of the human. Figure 3 illustrates an example of a 25 trip dial-a-ride problem. Rather than displaying order and structure, the problem resembles so much spaghetti. It is clear that for such problems the human interactor needs more help in generating good columns for the set partitioning model.

Unfortunately, it is also more difficult to apply straight forward methods such as the savings approach discussed earlier for the delivery example. In generating a dial-a-ride route one must be concerned with where both the origin and the destination occur in the sequence in order to calculate the potential savings. In addition, the capacity constraint may negate what otherwise appears to be good positions for the origin and destination in the sequence.

Because of this complexity, it is helpful to "decompose" the problem into two levels which we call "clustering" and "chaining". In essence, we consider a route to be made up of two components. Clusters correspond to trips which can all be on a vehicle at one time, while chains correspond to movement from the end of one cluster to the beginning of the next. Figure 4 provides an example of five good clusters, while Figure 5 illustrates one way in which these five clusters might be linked into two chains.

The partitioning model and pricing concepts can be effectively exploited

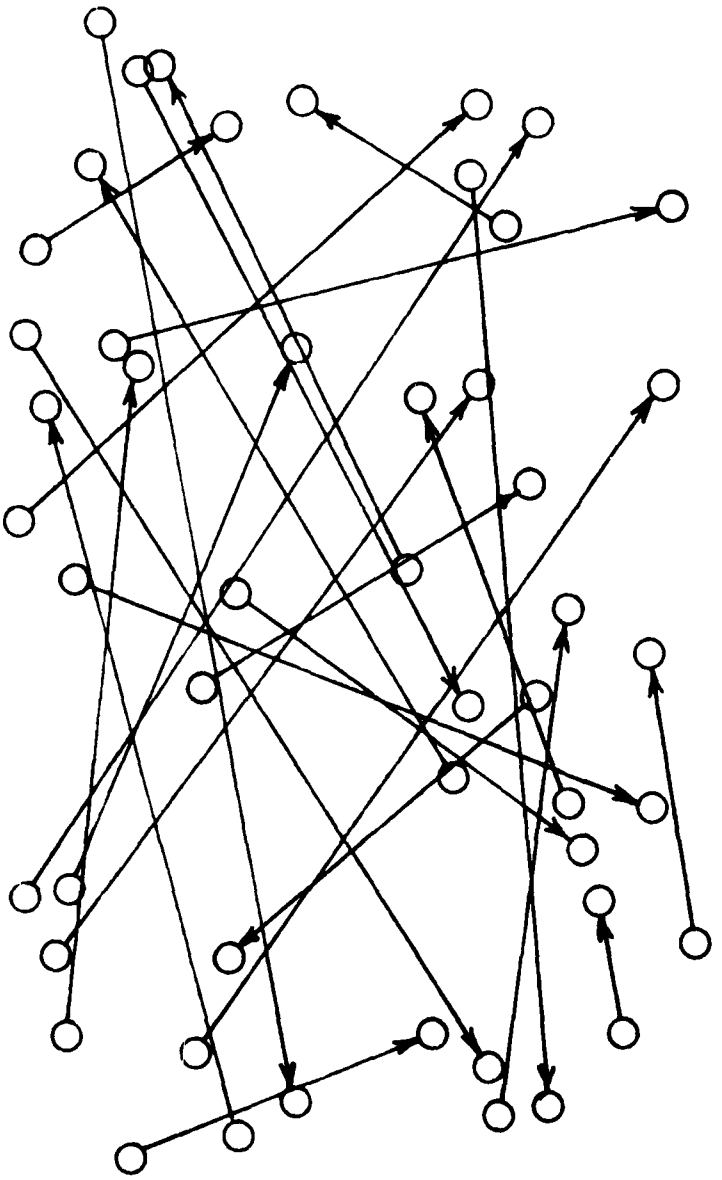


Figure 3. An Example of a 25 Trip Dial-a-Ride Problem

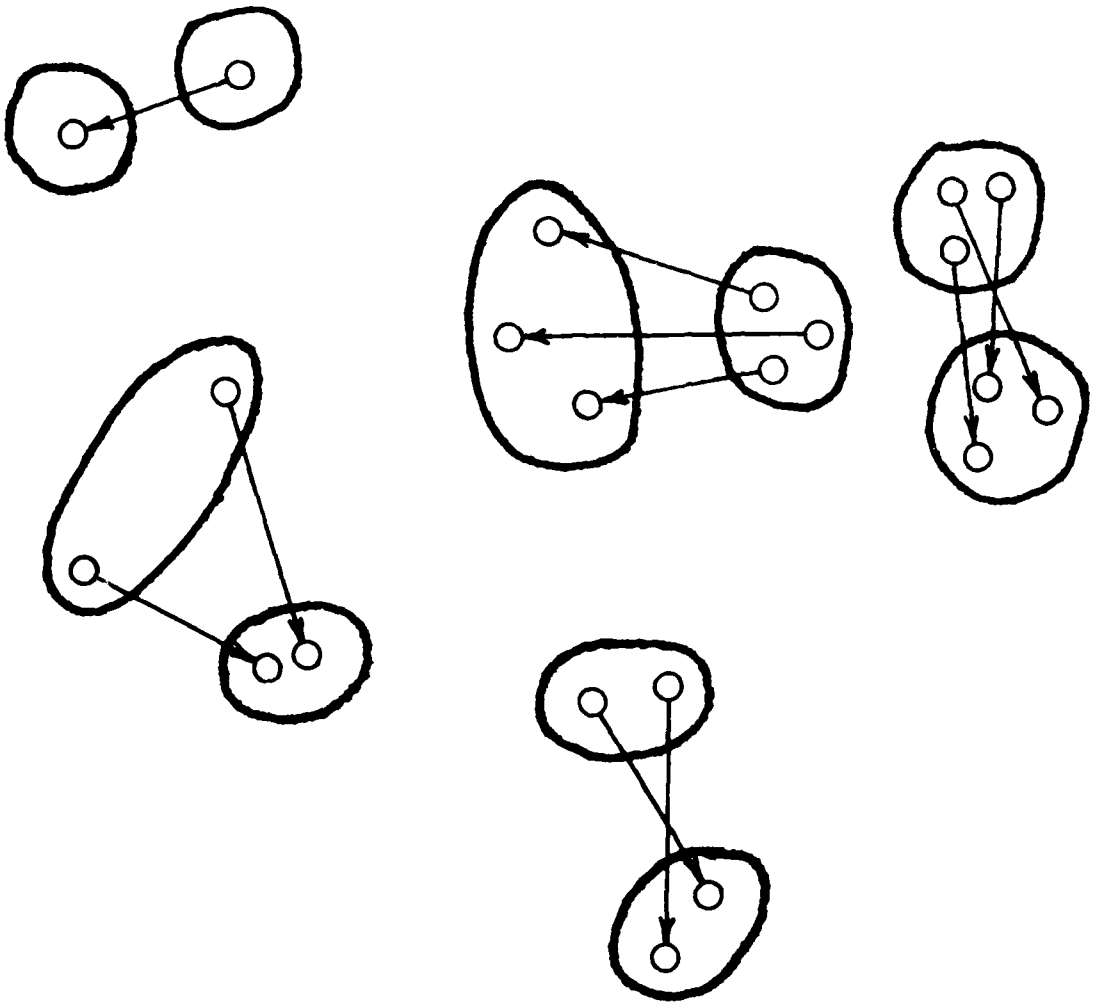


Figure 4. Example of Five Clusters

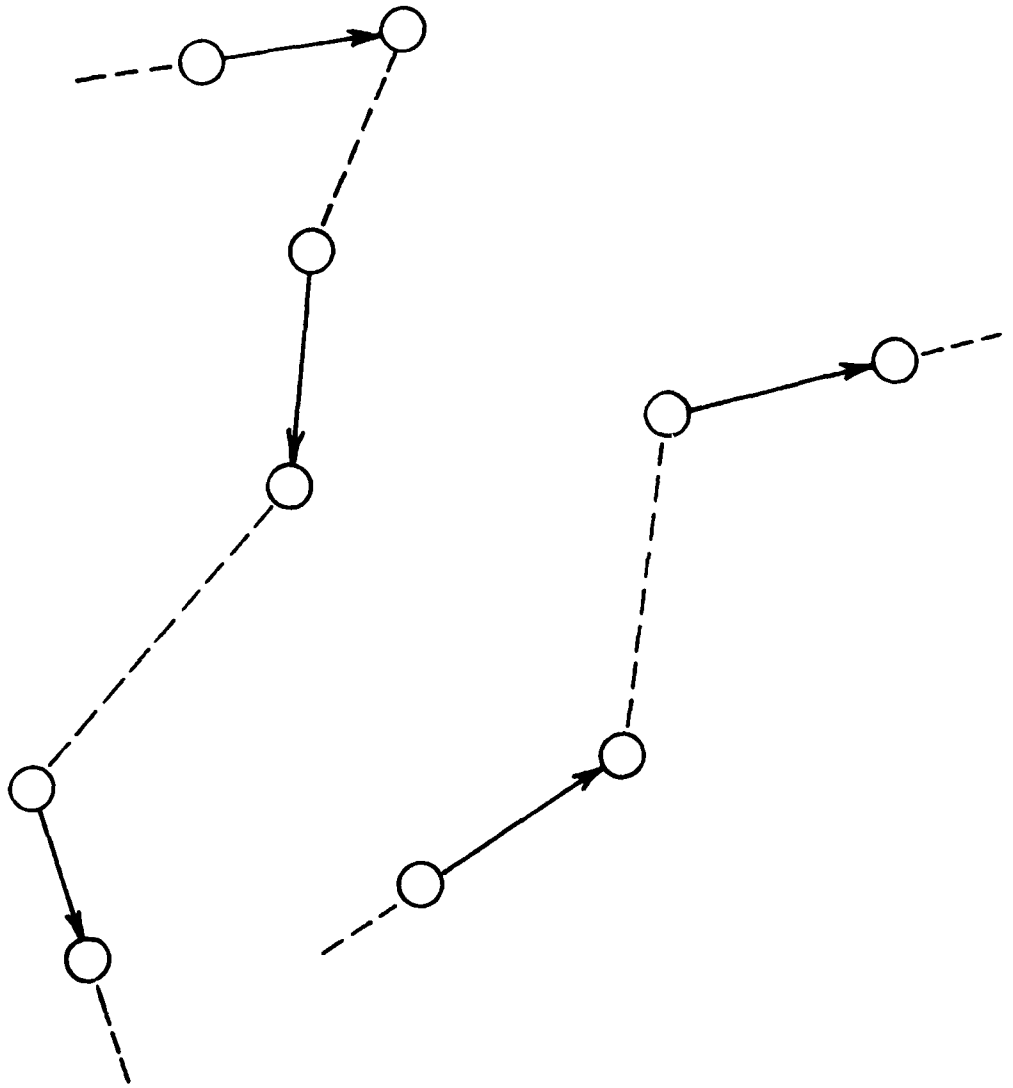


Figure 5. Example of Two Chains Linking the Five Clusters of Figure 4

to aid in generating improving clusters and chains in a column generation approach to solving two-ended vehicle routing problems. One partitioning model can be utilized in generating improving clusters while another partitioning model helps identify better chains.

In the partitioning model for clustering, the columns represent clusters and the rows represent the individual trips (demand for service). We shall demonstrate, in the next section, just how the pricing information from the partitioning problem can be used to generate additional clusters.

In the partitioning model for chaining, the columns represent chains and the rows represent the individual trips. The function of the chaining partitioning model is to combine the clusters into good vehicle routes.

The overall solution procedure consists of linking the clustering models and chaining models together in an interactive manner. The clustering models, partitioning matrix and pricing information are used to generate good clusters. These clusters are then passed to the chaining phase where they are linked together via the chaining models, partitioning matrix and associated pricing information. After chaining, it is possible to return to the clustering phase to identify additional clusters.

The next two sections discuss the specifics of clustering and chaining

Clustering

In the previous section we introduced the clustering concept. In this section we shall provide more details of the concept as well as the structure and operation of various clustering models. Figure 6 depicts a typical cluster (in this case, three trips).

Clustering makes sense if the origins are reasonably close together and the destinations are also close together. One way to

develop an evaluation of such circumstance is to locate the centroid of the origins, the centroid of the destinations and accumulate the resulting distances from the original trips. In Figure 6 we could evaluate the distances represented by (a+b+c) + (d+e+f). If this sum is small then it would make sense to cluster the trips.

By utilizing surrogate distances we lose the actual route distance evaluation; however, we gain the ability to evaluate large numbers of cluster possibilities conveniently and simultaneously. In Figure 6 we might employ Euclidean distances. In this case we could compare the sum of the row prices, $p_1 + p_2 + p_2$, generated in the covering model for clustering to the quantity 2(a+b+c) + 2(d+e+f) + g to determine whether clustering is appropriate. We can think of the latter quantity as a surrogate for the vehicle routing distance.

We can develop a straight forward extension of the Location - Allocation model discussed in Section 7 to identify good clusters for the dial-a-ride problem. In place of the quantity

$$\sum_i \sum_j 2[(x_j - a_i)^2 + (y_j - b_i)^2]^{\frac{1}{2}} z_{ij}$$

we would employ a quantity

$$\sum_i \sum_j 2[(\bar{x}_j - \bar{a}_i)^2 + (\bar{y}_j - \bar{b}_i)^2] z_{ij}$$

$$+ \sum_i \sum_j 2[(\hat{x}_j - \hat{a}_i)^2 + (\hat{y}_j - \hat{b}_i)^2]^{\frac{1}{2}} z_{ij}$$

where $\bar{x}_j$, $\bar{y}_j$, $\bar{a}_i$ and $\bar{b}_i$ correspond to origins and $\hat{x}_j$, $\hat{y}_j$, and $\hat{a}_i$ $\hat{b}_i$ correspond to destinations of trips.

It is also possible to develop a savings approach to clustering in a similiar manner to that described earlier for the delivery problem.

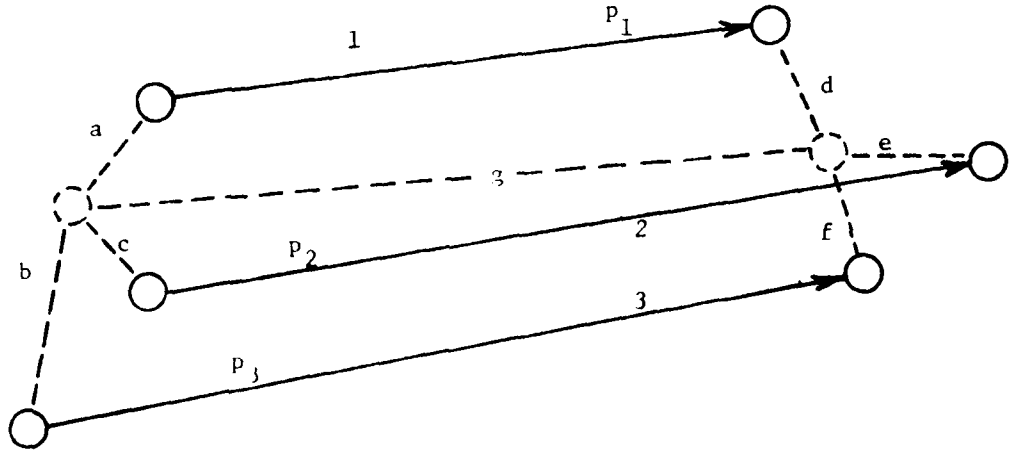


Figure 6. Surrogate Distances for a Typical Cluster

Chaining

Upon termination of the clustering process, a number of reasonably good clusters are available. This set includes not only the "best" cluster sets as selected by the covering model for clustering, but also a number of other clusters which might be nearly as good, but which were not selected in the optimal solution to the covering model. Figure 7 illustrates a set of clusters which might result from the clustering process. The clusters in the figure represent only the optimal solution to the clustering process. In addition, we might have other clusters available, e.g., a cluster containing only trips 1 and 2.

The clusters obtained represent good possibilities for segments (legs) of a vehicle route. The next step in the process is to link ("chain") these clusters into complete vehicles routes. Figure 8 illustrates the chaining concept. Trips 1 and 2 form one cluster, while trips 3, 4 and 5 form another cluster (see Figure 8a). In Figure 8b, trips 1 and 2 are replaced by a single pseudo (cluster) trip, as is also the case for trips 3, 4 and 5. These cluster trips are then chained together.

The interpretation of chaining is that a single vehicle will service the first set of trips (in Figure 8b these would be trips 1 and 2) and then proceed to service the next set of trips (i.e., trips 3, 4 and 5) in the chain. Figure 9 illustrates the likely vehicle route to service the two clusters.

We shall demonstrate how mathematical models can be developed to aid the human in developing good chains (vehicle routes).

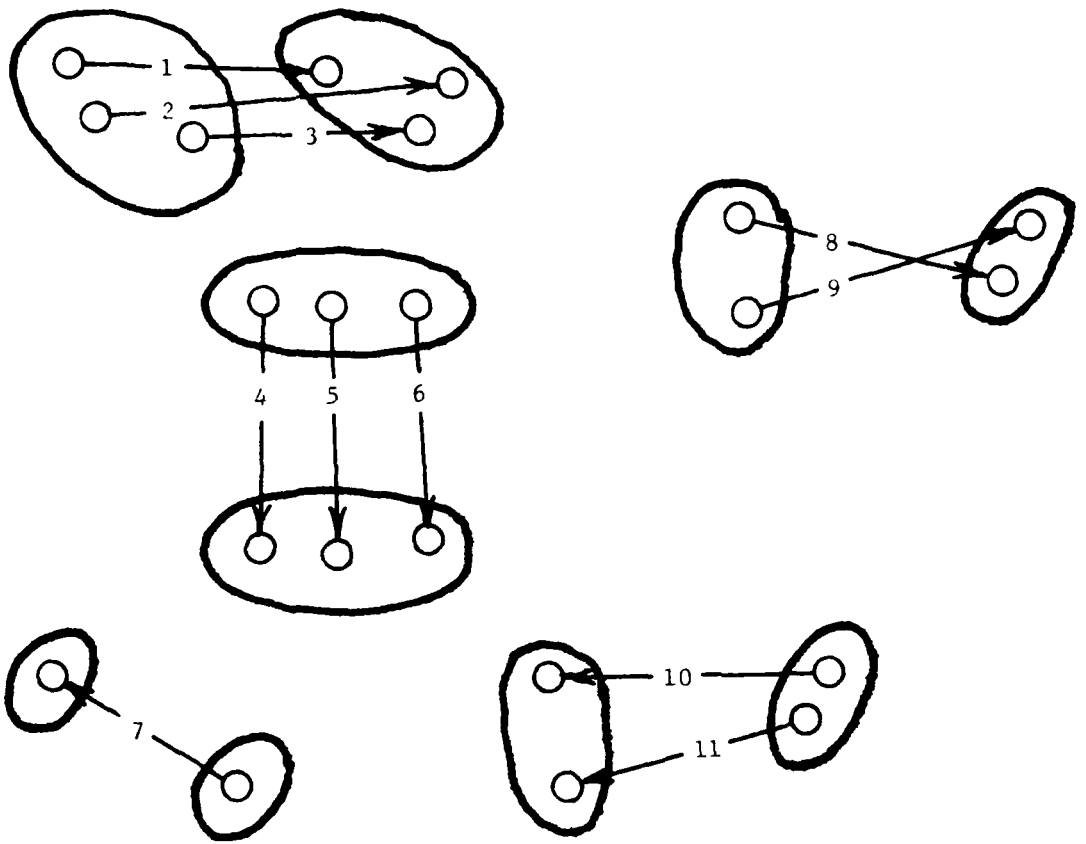
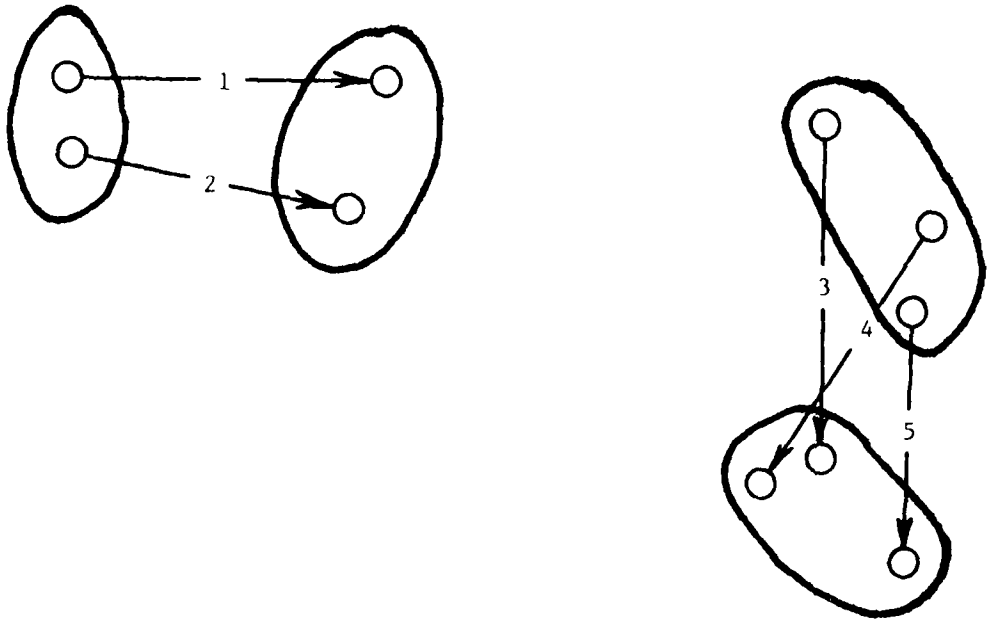
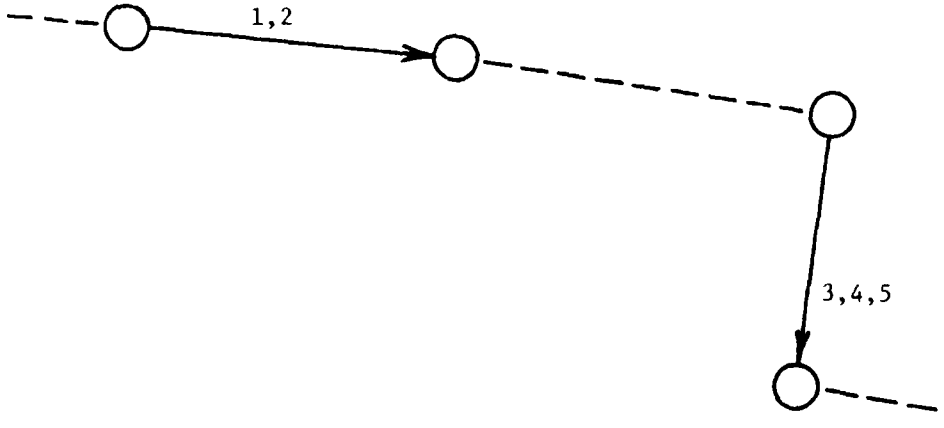


Figure 7. Example of Clusters Resulting from the Clustering Process



a. Two Typical Clusters



b. The Associated Cluster Arcs Chained Together

Figure 8. An Example of Chaining Clusters Together

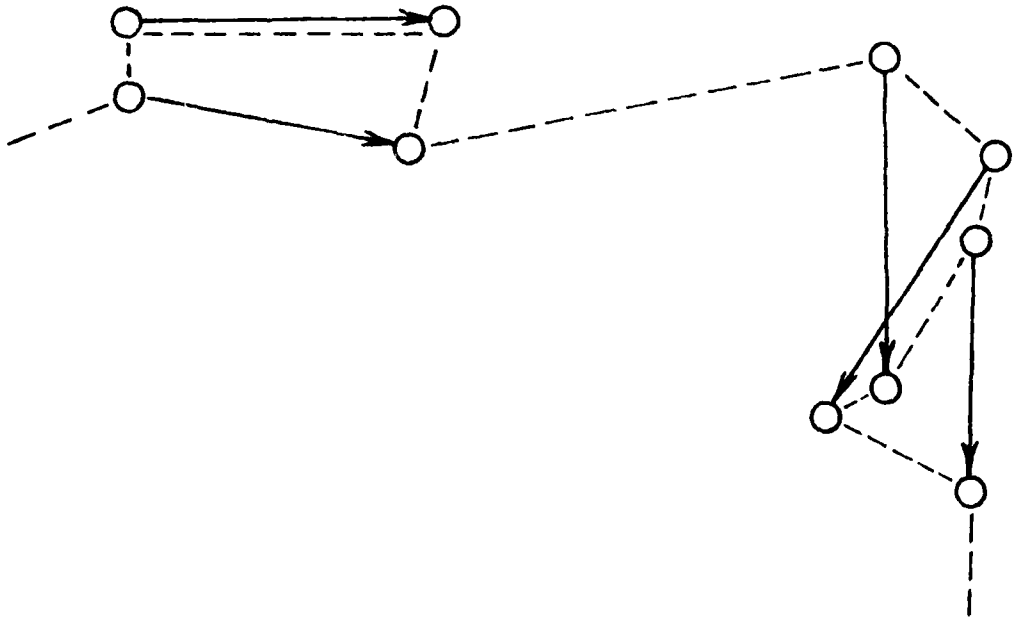


Figure 9. Probable Vehicle Route for the Chain of Figure 8b.

The most fundamental model of chaining is based on network flows. In a network we represent each cluster trip by two nodes (an origin and a destination node) and an arc. We then add additional arcs to the network which link the destination node of one cluster arc to the origin node of another cluster arc on the basis of proximity considerations. In Figure 10, we indicate one network flow model for the clusters of Figure 7.

To establish the flow variables for the network flow problem consider the case where each trip appears in exactly one cluster. We assign every arc an upper capacity of 1, the dashed arcs a lower capacity of 0 (zero) and the solid cluster arcs a lower capacity of 1 indicating that these trips must be serviced. Arc flow costs for the network flow problem are the associated vehicle travel distances. If the "starting node" and the "ending node" represents the vehicle storage depot, then we may also assign additional costs to the arcs originating at the starting node to reflect the fixed cost of using each vehicle (provided they are all the same).

If the network flow problem for chaining contains no circuits and if the same trip does not appear in two different clusters in the network, then the resulting chains (vehicles routes) will be valid ones. If either of these two conditions are explicitly included in the model, then the underlying network flow problem becomes considerably more complicated.

We can circumvent some of the problems associated with the difficult problems of chaining by utilizing a partitioning model of chaining. (Actually we substitute one difficulty for another.) In a partitioning model of chaining,

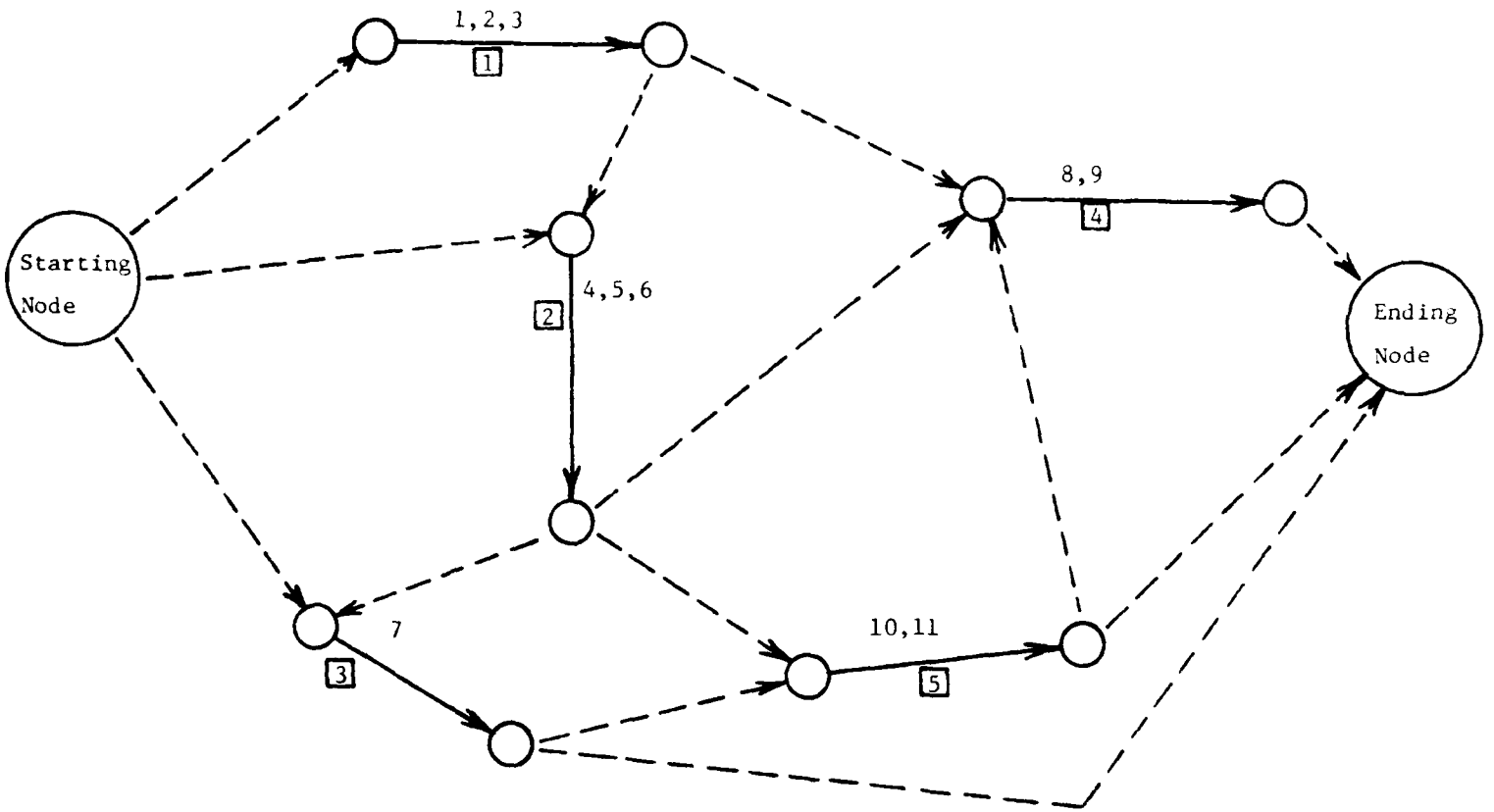


Figure 10. The Network Flow Problem for Chaining of the Clusters in Figure 7

we associate a column of the partitioning matrix for each feasible chain (vehicle route) and a row for each trip. Column j contains 1 in row i if trip i is serviced by chain (vehicle route) j. Otherwise, it contains a 0 (zero). The zero-one variable associated with the columns provide indications of which chains were selected in the optimal partitioning solution. Table 3 illustrates a partitioning matrix for several chains in Figure 10.

Besides solving the partitioning model optimally (which we would not likely do), the obvious difficulty with such a model is in generating the candidate chains (columns). To aide in this column generation process, we could employ (1) the human (which would certainly be part of any interactive process), (2) a savings approach to combine clusters into chains, (3) a network flow model or (4) some other method. The first two of these column generation approaches should be reasonably intuitive to develop. We shall briefly describe the third one. Suppose we have identified some candidate columns (chains) for the partitioning problem (e.g. the individual clusters from the clustering process). Then, the partitioning model will yield (1) an optimal solution and (2) a set of row prices. As before, a column j not in the solution appears favorable if

$$\sum_i p_i a_{ij} - c_j > 0.$$

Table 3. An Example Covering Model for Certain Chains in Figure 10.

	clusters 1 & 2	clusters 1 & 4	clusters 2 & 5	clusters 1, 2 & 5	clusters 3, 4 & 5	clusters 1, 2 & 4	clusters 1, 2, 3 & 5
	1	2	3	4	5	6	7
1	1	1		1		1	1
2	1	1		1		1	1
3	1	1		1		1	1
4	1		1	1		1	1
5	1		1	1		1	1
6	1		1	1		1	1
7					1		1
8		1			1	1	
9		1			1	1	
10			1	1	1		1
11			1	1	1		1

Now, c_j is the cost (distance) of servicing the chain. The $\sum_i p_i a_{ij}$ is the cost of servicing the given trips in the current partitioning solution. Consider how these two terms might be represented in the network of Figure 10.

Suppose we associate with each dashed arc the negative of whatever costs that are incurred in traversing the arc (this is the same as before with a sign change). Associated with each clustering arc the quantity $\sum p - c$, where the term $\sum p$ represents the sum of row prices for trips in the cluster and c represents the cost of the cluster. With these costs defined on the network, we seek a path (or paths) in the network, from the starting to the ending node, which has positive total cost. It should be clear that such a path satisfies the condition for a potentially improving chain, and can be added to the partitioning problem.

Again, with this network flow model (a shortest path model for a single chain) we have the difficulties associated with circuits and with the same trip serviced several times by a single chain. The phenomenon of a trip being serviced several times seems to occur infrequently since in seeking a least cost solution, the model tends to avoid such an instance. When it does occur, an attractive heuristic is to simply delete one of the conflicting clusters from the column being generated. We are currently utilizing the human interactor to handle the case where the flow problem contains circuits. The human breaks the circuits and patches the paths back together. If we re-

strict the procedure to locating a single improving path each time, the resulting shortest path model lends itself more intuitively to the development of good heuristic procedures for solving both of the major difficulties associated with the flow mode. In particular it is much easier for the human to break the circuits in a single path than in multipaths.

Conclusions

An interactive dial-a-ride system based on the concepts presented here has been implemented on a Chromatics color graphics terminal interfaced with a CYBER 74 mainframe computer. Although the system is in many ways very rudimentary, it dramatically indicates the potential for this kind of interactive optimization. We are in the process of making extensive modifications in the software and are testing a variety of new models and heuristics to aid in route generation.

References

1. Balas, E. and Padberg, M. W., "Set Partitioning: A Survey." SIAM Review, 18, (1976) 710-760.

2. Balinski, M. L. and Quandt, R. E., "On An Integer Program for a Delivery Problem," Operations Research, 12 (1964) 300-304.

3. Clarke, G. and Wright, J. W., "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," Operations Research, 12 (1964) 569-581.

4. Cooper, L., "N-Dimensional Location - Allocation Used for Cluster Analysis," Report No. C00-1493-23, Washington University, St. Louis, MO, 1969.

5. Krolak, P. D. and Nelson, J. H., "A Family of Truck Load Clustering (TLC) Heuristics for Solving Vehicle Scheduling Problems," Technical Report 78-2, Computer Science, Vanderbilt University, Nashville, TN, 1978.

IV. ORDER PICKING PROBLEM

One of the most fundamental, among the many important item retrieval problems associated with warehousing and materials handling is what we will call the simple "order-picking problem." An order consists of a subset of the items stored in a warehouse. When an order is requested, a vehicle is dispatched from the shipping area to collect or "pick" the items in the order and transport them back to the shipping area. The objective is to minimize distance traveled by the vehicle. We will assume that a vehicle picks only one order at a time and that an order does not exceed the vehicle's capacity.

For an arbitrary aisle configuration within a warehouse, the order-picking problem is easily recognized as a variant of the well known and difficult to solve traveling salesman problem. Fortunately, the most common warehouse aisle configuration is that given in Figure 1. For this configuration we will present an algorithm for the order-picking problem which is linear in the number of aisles.

Graph Representation

Consider an order containing m items which is to be picked in a warehouse with the aisle configuration illustrated in Figure 1. Define a graph G by associating a vertex v_0 with the shipping area location, a vertex v_i with the location of each item $i = 1, 2, \ldots, m$ in the order, and vertices a_j and b_j with the ends of each aisle $j = 1, 2, \ldots, n$. Connect any two vertices in G which correspond to adjacent locations in the warehouse by an unlimited number of parallel arcs, each with length equal to the direct distance between the two locations. An example graph is given in Figure 2 corresponding to the warehouse and order

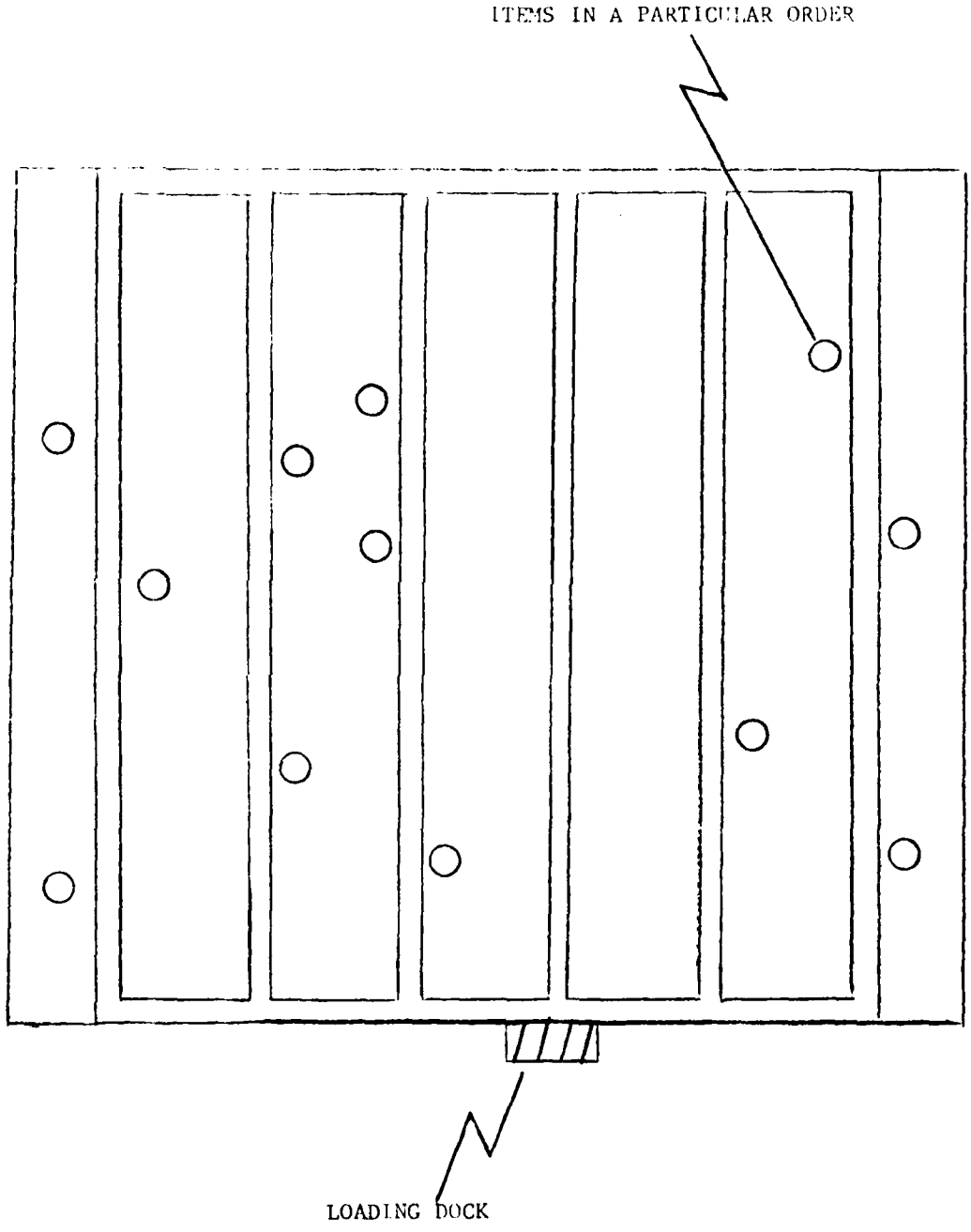


Figure 1: Warehouse Aisle Configuration

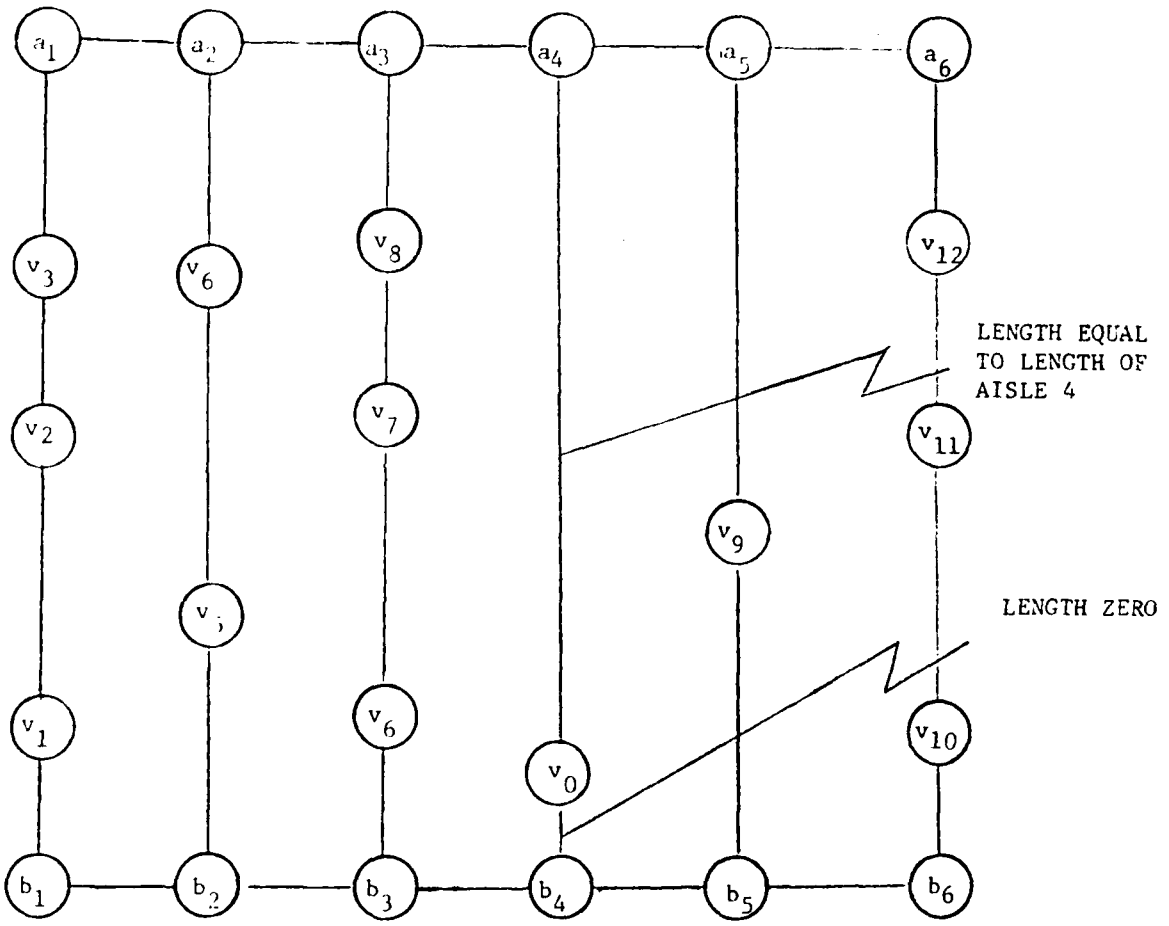


Figure 2: Order-Picking Graph Where Each Arc Represents An Unlimited Number of Parallel Arcs

depicted in Figure 1. For simplicity the parallel arcs connecting adjacent vertices have been represented by a single arc. Also for simplicity v_0 is depicted as being between vertices A_4 and B_4 instead of coinciding with B_4.

An order picking tour is a cycle in G which includes each of the vertices v_i for $i = 0, 1, \ldots, m$ at least once. Note that since each adjacent pair of vertices is connected by an unlimited number of parallel arcs, we can assume without loss of generality that an order-picking tour contains any arc at most once. The order-picking problem is then to find a minimum length order-picking tour in G.

A subgraph $T \subset G$ will be called a tour subgraph if there is an order-picking tour which uses each arc in T exactly once. The following characterization of a tour subgraph is a specialization of a well known theorem on Euler graphs.

Theorem 1. A subgraph $T \subset G$ is a tour subgraph if and only if (a) T contains all vertices v_i for $i = 0, 1, 2, \ldots, m$; (b) T is connected; and (c) every vertex in T has even degree.

Given a tour subgraph, we will show in a later section that an order-picking tour can be very efficiently determined. Hence, by developing an efficient procedure for finding a minimum length tour subgraph we can efficiently solve the order-picking problem. The following corrollaries of Theorem 1 are useful in characterizing a tour subgraph.

Corrollary 1. A minimum length tour subgraph contains no more than two arcs between any pair of vertices.

Corrollary 2. If $(P, \bar{P})$ is any node partition of a tour subgraph, there is an even number of arcs with one end in P and the other in $\bar{P}$.

Let L_j be a subgraph of G consisting of vertices a_j and b_j and everything in G to the left of a_j and b_j but not including arcs and vertices between or to the right of a_j and b_j. An illustration is given in Figure 3. A subgraph $T_j \cup L_j$ is an L_j-partial tour subgraph if it contains every $v_i \in L_j$ and at least one of $\{a_j, b_j\}$, every vertex in T_j except possibly for a_j and b_j have even degree, and T_j has either a single connected component or two connected components with a_j in one component and b_j in the other. Some examples illustrating L_5-partial tour subgraphs are given in Figure 4.

Two L_j-partial tour subgraphs T_j^1 and T_j^2 are equivalent if a_j has the same degree in both, b_j has the same degree in both, and either both have one connected component or both have two connected components.

Theorem 2. If T_j^1 and T_j^2 are two equivalent L_j partial tour subgraphs, and R is a subgraph of $G-L_j$ such that $T_j \cup R$ is a tour subgraph, then $T_j^2 \cup R$ is also a tour subgraph.

Proof. Since $T_j^1 \cup R$ is a tour subgraph, R must contain all $v_i \in G-L_j$ and except possibly for a_j and b_j, each vertex in R must have even degree. Hence, $T_j^2 \cup R$ contains all $v_j \in G$. Since a_j and b_j have the same degree in both T_j and T_j^2 and all other vertices in T_j^2 are even, all vertices in $T_j^2 \cup R$ must have even degree.

Now note that shrinking a connected component of a graph to a single vertex does not affect the connectivity of the graph. If in $T_j^1 \cup R$ and $T_j^2 \cup R$ we shrink the connected components of T_j^1 and T_j^2 to single vertices the resulting graphs are identical (i.e., either T_j^1 and T_j^2 each shrink to a single vertex connected to a_j and b_j in the same fashion or they each shrink to two vertices connected to a_j and b_j in the same fashion). Therefore since $T_j^1 \cup R$ is connected $T_j^2 \cup R$ is also connected. It then follows from Theorem 1 that T_j^2 is a tour subgraph.

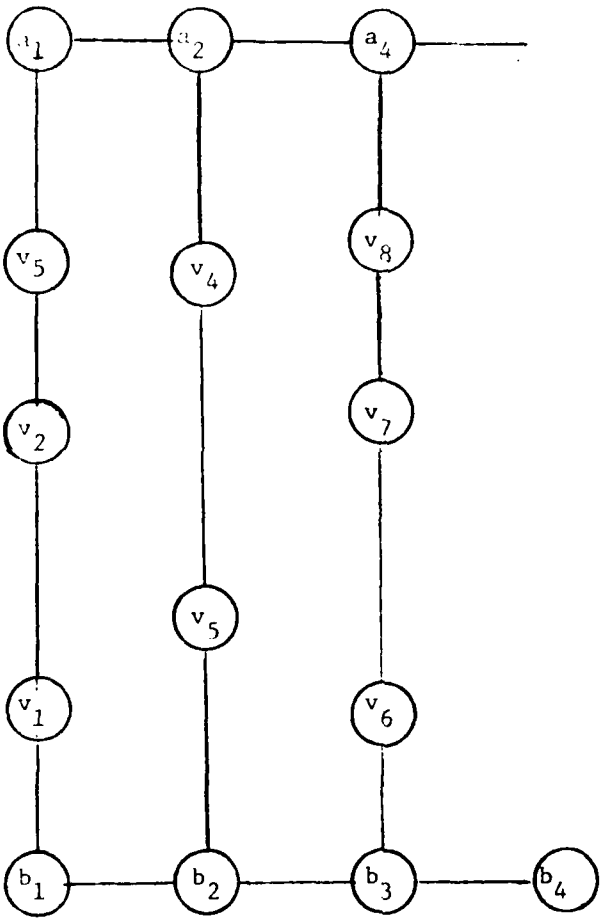


Figure 3: An L_4 Subgraph Of The Order-Picking Graph In Figure 1

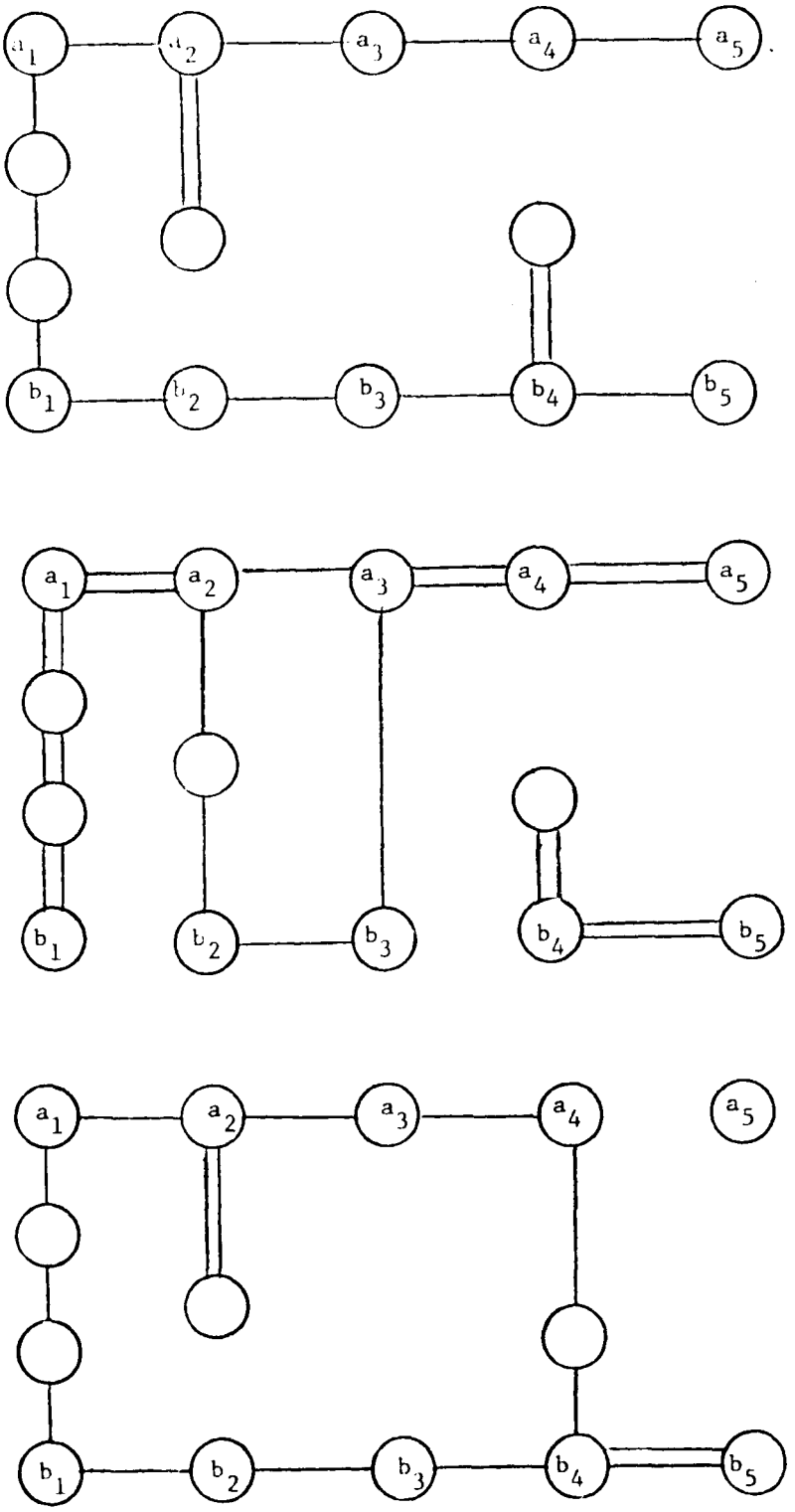


Figure 4: Examples of L_5-Partial Tour Subgraphs

Constructing Minimum Length Tour Subgraphs

Before constructing a minimum length tour subgraph, we can simplify the order-picking graph somewhat by use of the following result.

Lemma 1: If an aisle does not contain at least one item in a given order then there is an optimum order-picking tour which does not traverse the aisle.

Proof: Consider any two vertices v_i and v_j which are adjacent in an optimum tour (see Figure 5). The minimum distance traveled between v_i and v_j is D = min {a+c,b+d} + h. Clearly, this can be attained without traversing any aisle other than those containing v_i and v_j.

Therefore we can delete from consideration those arcs corresponding to aisles which contain no items in the order. For simplicity, in the following discussion we will assume that this has been done and the aisles and corresponding vertices renumbered.

Consider any minimum length tour subgraph T and the vertex pair a_j,b_j corresponding to the ends of some aisle j. From Corrollary 1 we know that the degree of every vertex in T is even and from Corrollary 2 we know that for any $j > 1$ the number of arcs connecting a_{j-1} and a_j plus the number of arcs connecting b_{j-1} and b_j is also even. Eliminating all possibilities which do not satisfy these conditions, we are left with the 80 cases in Figure 6 for the connectivity of a_j and b_j in T. Vertices between a_j and b_j have been ignored except where they are necessary to indicate connectivity.

From Theorem 1 the degree of each vertex in T must be even. This eliminates the cases 0_1 through 0_{34} which contain odd vertices. The vertices a_j and b_j represent intersections but not elements in the

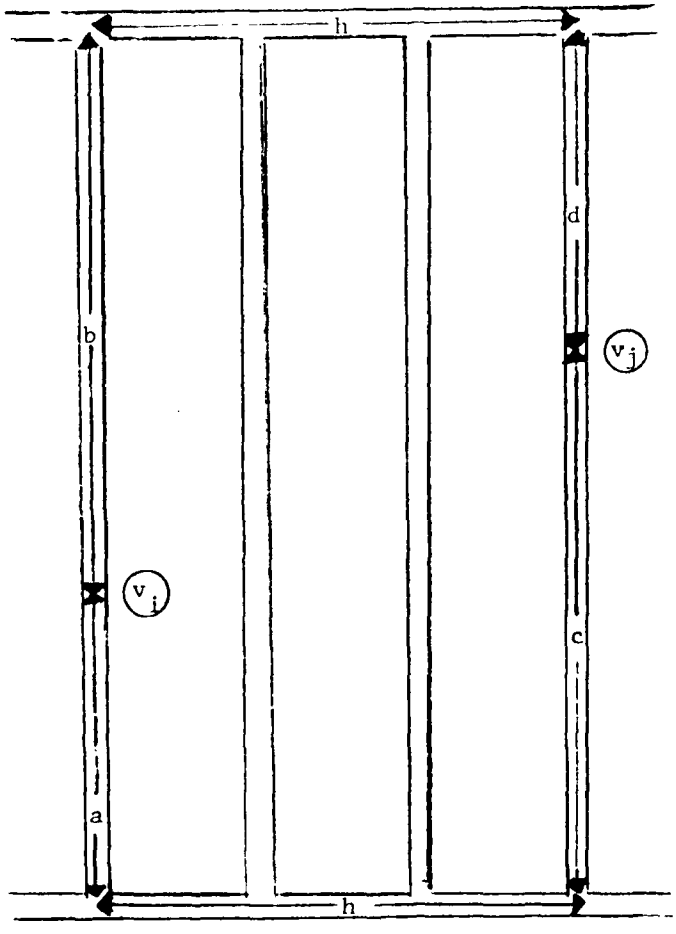


Figure 5: Distance Traveled When v_i And v_j Are Adjacent In An Order-Picking Tour

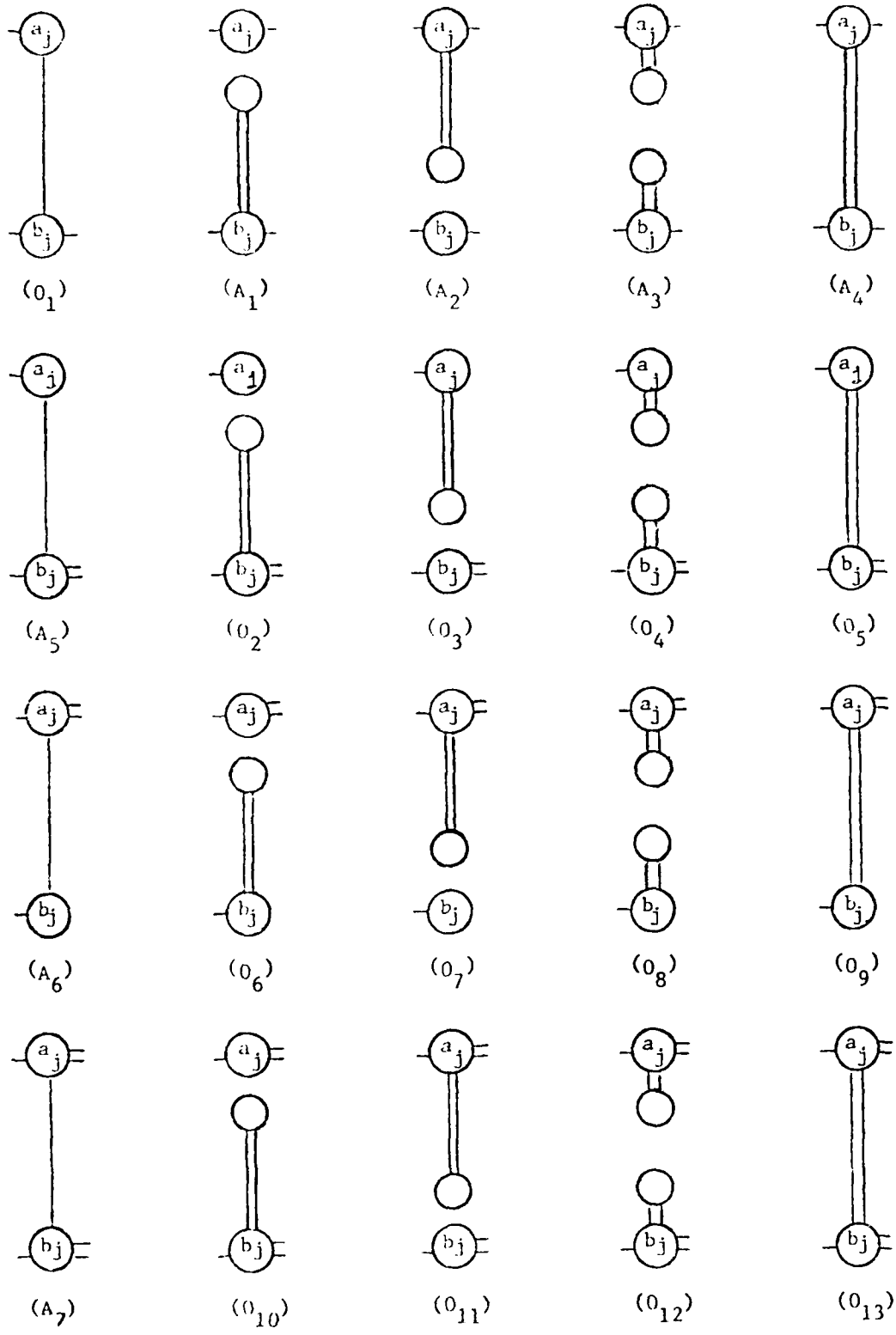


Figure 6: Possibilities For Connectivity Of a_1 And b_1 For $j = 2,3,\ldots,n-1$

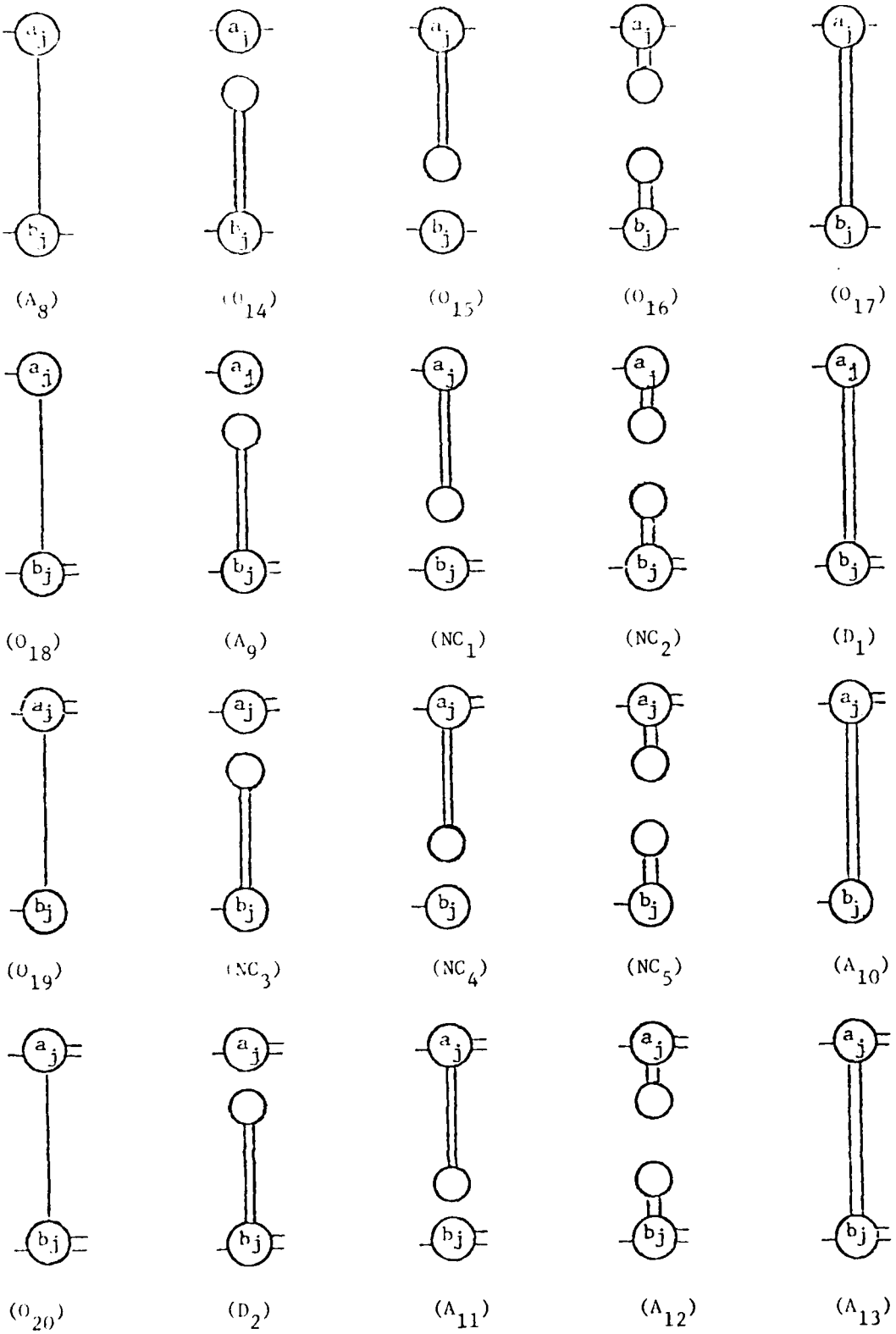


Figure 6: Continued

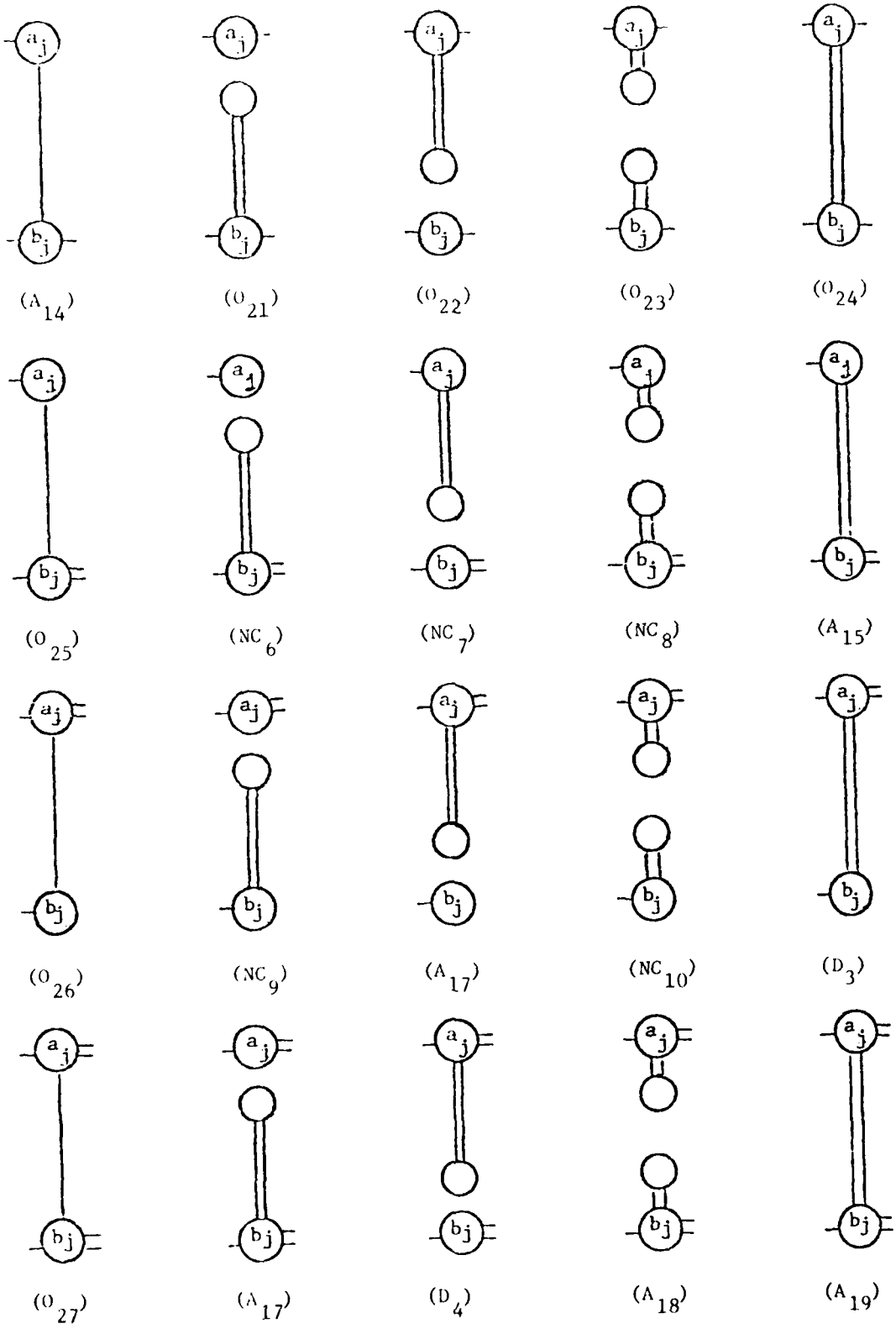


Figure 6: Continued

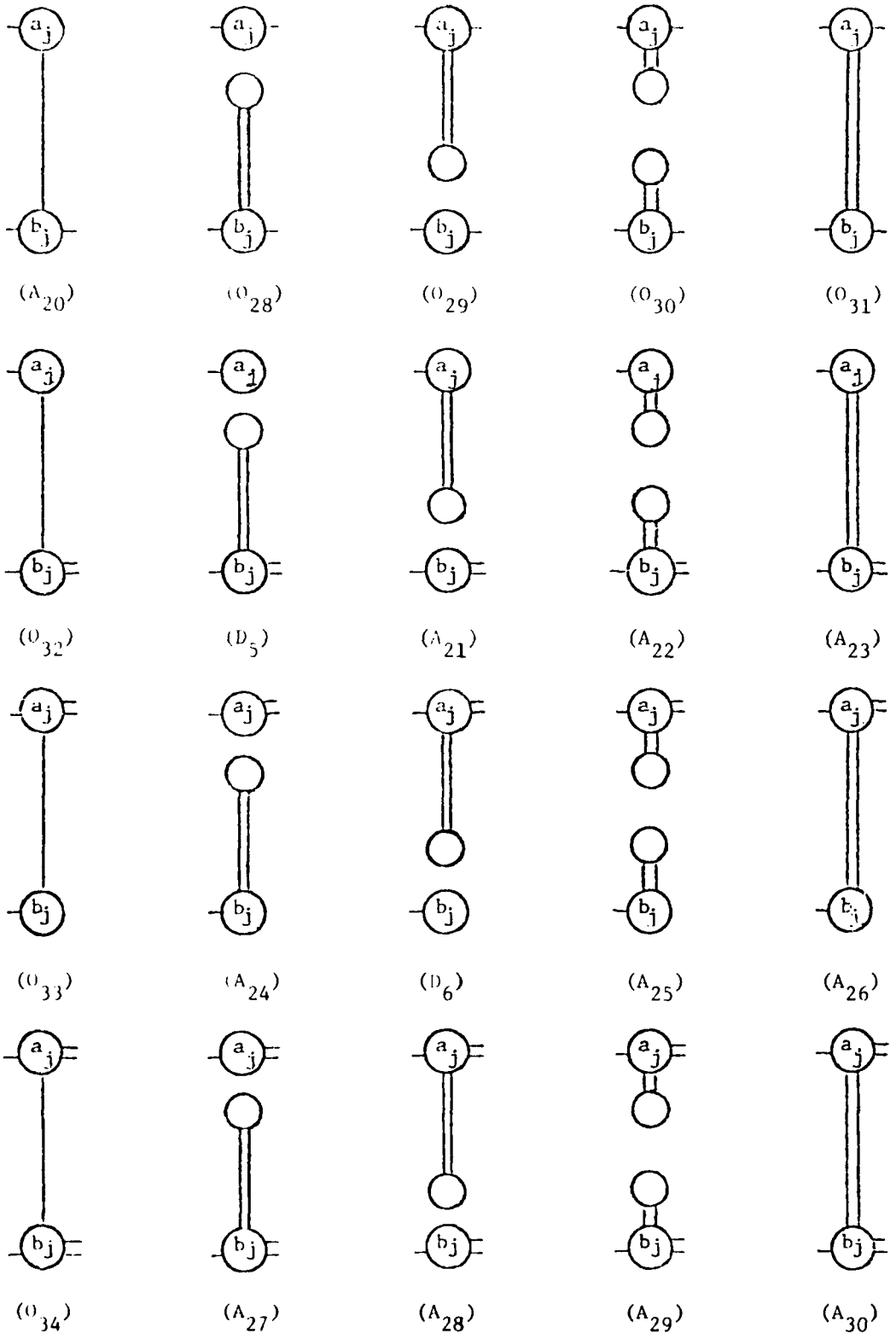


Figure 6: Continued

order. Hence, we can eliminate cases D_1 through D_6 since they would never appear in an optimum tour. For example, case A_9 dominates cases D_1, D_2, and D_5 since for those cases we do not need to visit a_j. Similarly case A_{17} dominates cases D_3, D_4, and D_6.

Note that between vertices a_j and b_j the tour subgraph takes one of the forms in Figure 7. For (a), (b), (c), and (e) the distance traveled within the aisle is unique. For an aisle containing p items to be picked, there are p-1 possibilities corresponding to case (d). However, the only possibility that we need to consider is determined by the two adjacent items within the aisle which are fartherest apart. Assuming p items to be picked in the aisle, this requires p-1 comparisons. We will assume that this has been done for each aisle. Hence, the length of that part of the tour between a_j and b_j is known for each of the arcs A_1 through A_{30}.

From Figure 6 we see that for i = 2,3,...,n-1 there are only four degree combinations for a_j and b_j in a L_j-partial tour subgraph which can lead to a tour subgraph. Denoting these combinations by (degree a_j, degree a_j) they are (1,1), (0,2), (2,0), and (2,2). If we examine the connectivity of each of these combinations, it follows from the definition that the (0,2) and (2,0) L_j-partial tour subgraphs must have only one connected component. The (1,1) L_j-partial tour subgraphs must also have only one connected component since to have two components with one connected to a_j and the other to b_j implies that some vertex other than a_j and b_j has odd degree. Finally, the (2,2) L_j-partial tour subgraphs can have either one or two components. Hence, there are five equivalence classes which we will denote (1,1), (0,2) (2,0), (2,2,1C), and (2,2,2C). Under our definition of "equivalent," all possible L_j-

partial tour subgraphs fall into one of these classes.

It follows from Theorem 2 that we only need to concern ourselves with the minimum length L_j-partial tour subgraph in each equivalence class in constructing a minimum length tour subgraph. To see how a minimum length L_j-partial tour subgraph can be constructed for each equivalence class, assume that for some $j = 3,4,\ldots,n-1$ we have minimum length L_{j-1}-partial tour subgraphs for each of the five equivalence classes.

First consider the minimum length L_{j-1}-partial tour subgraph corresponding to the (1,1) class. If we add the arcs indicated in Figure 6 case A_7 between a_{j-1} and b_{j-1}, between a_{j-1} and a_j, and between a_{j-1} and b_j, we will have constructed an L_j-partial tour subgraph in the (2,2,1C) equivalence class. Figure 7 indicates, in terms of the cases enumented in Figure 6 all of the possible transitions from a L_{j-1}-partial tour subgraph to a L_j-partial tour subgraph. Note that the cost of the transition is simply the length of the arcs added in going from an j-1 equivalence class to a j equivalence class. For situations such as the transition from (1,1) to (1,1) in Figure 7 the cost of the transition is the minimum of the cases A_1, A_2, A_3, A_4. Actually the A_4 case is dominated here as are A_{23}, A_{26}, and A_{30} for the (2,2,1C) to (0,2), (2,2,1C) to (2,0), and (2,2,1C) to (2,2,1C) transitions respectively. Hence, these cases can be deleted from consideration.

Therefore, to find the minimum length L_j-partial tour subgraph for each equivalence class, given the minimum length L_{j-1} partial tour subgraph, we first add the length of the L_{j-1} partial tour subgraph for each class to the transition cost. Then for each j equivalence class, we pick the

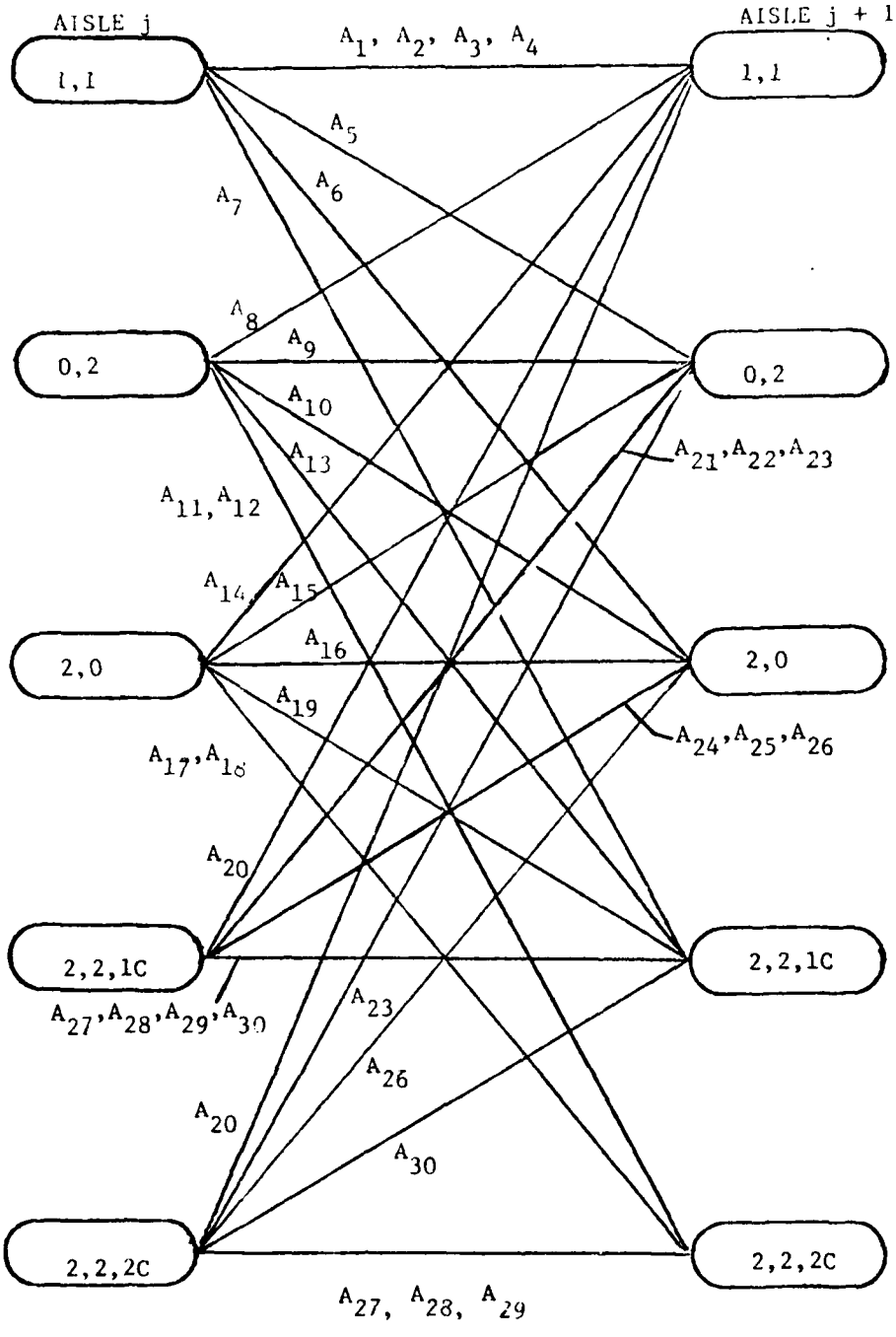


Figure 7: Possible Transitions From An L_{j-1}-Partial Tour Subgraph To An L_j-Partial Tour Subgraph

minimum of the sums which correspond to transitions to that class.

For $j = 1$ and $j = n$ the possible cases are shown in Figure 8. Aisle 1 has only a single equivalence class. The transition is shown in Figure 9. The cost of a minimum length L_2-partial tour subgraph for each equivalence class, is simply the transition cost. Hence we can easily start the process. Once a minimum length L_n-partial tour subgraph has been determined for each equivalence class, the transitions indicated in Figure 10 take us to a tour subgraph.

We can couch this procedure in dynamic programming terms or equivalently consider it a shortest path problem in an acylic graph (i.e. direct all of the arcs from lower to higher numbered aisles) where the vertices in Figures 7, 9, and 10 correspond to states.

Tour Construction

We have developed a procedure for finding a minimum length tour subgraph T from the order-picking graph. There remains the question of how to construct a tour from T. The following is a very simple and efficient procedure for constructing an optimum order-picking tour from T.

Step 1. Begin the tour by letting v_0 be the first vertex visited.

Step 2. Let v^* be the vertex currently being visited.

Step 3. If there is a pair of unused parallel arcs in T incident to v^*, use one of them to get to the next vertex. Go to Step 2.

Step 4. If there are any unused single arcs in T, (i.e., not one of a pair of parallel arcs) use one of them to get to the next vertex. Go to Step 2.

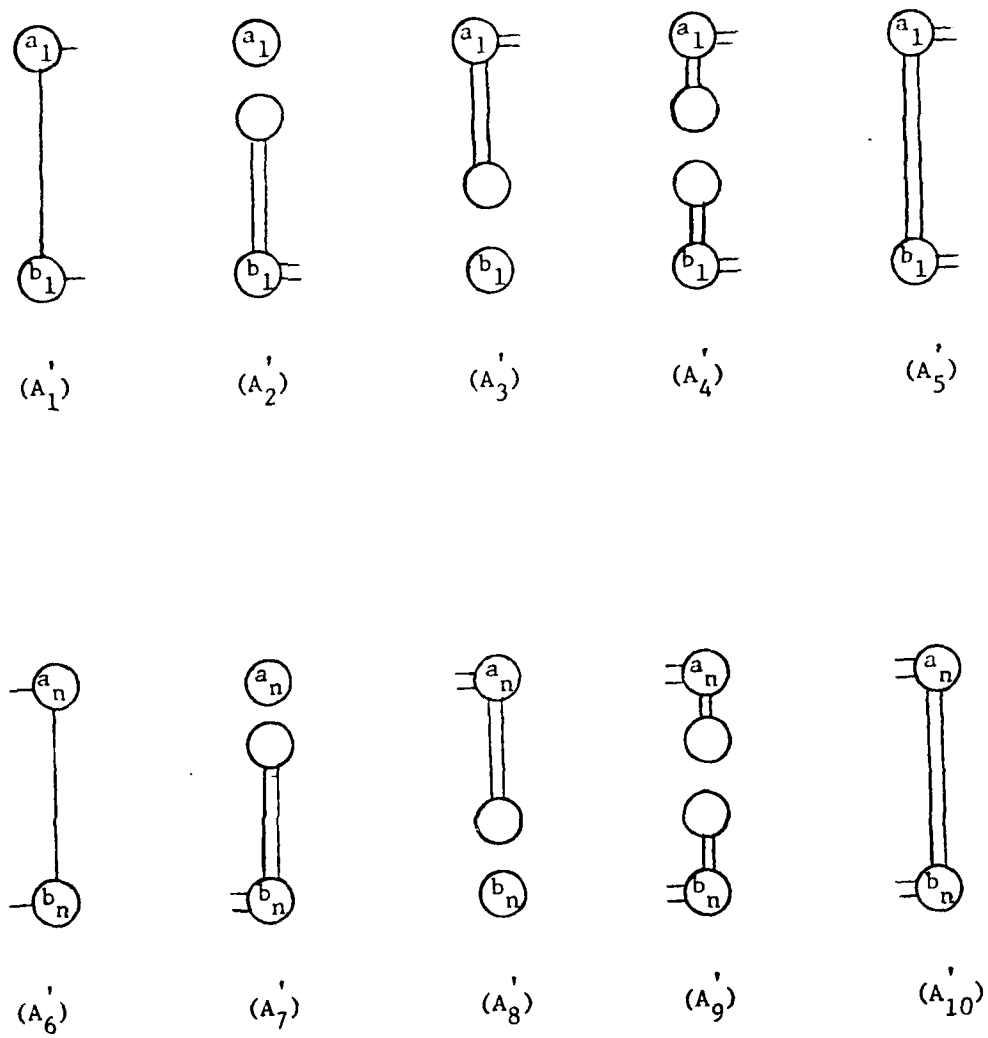


Figure 8: Possibilities For Connectivity Of a_1 And b_1 And a_n and b_n.

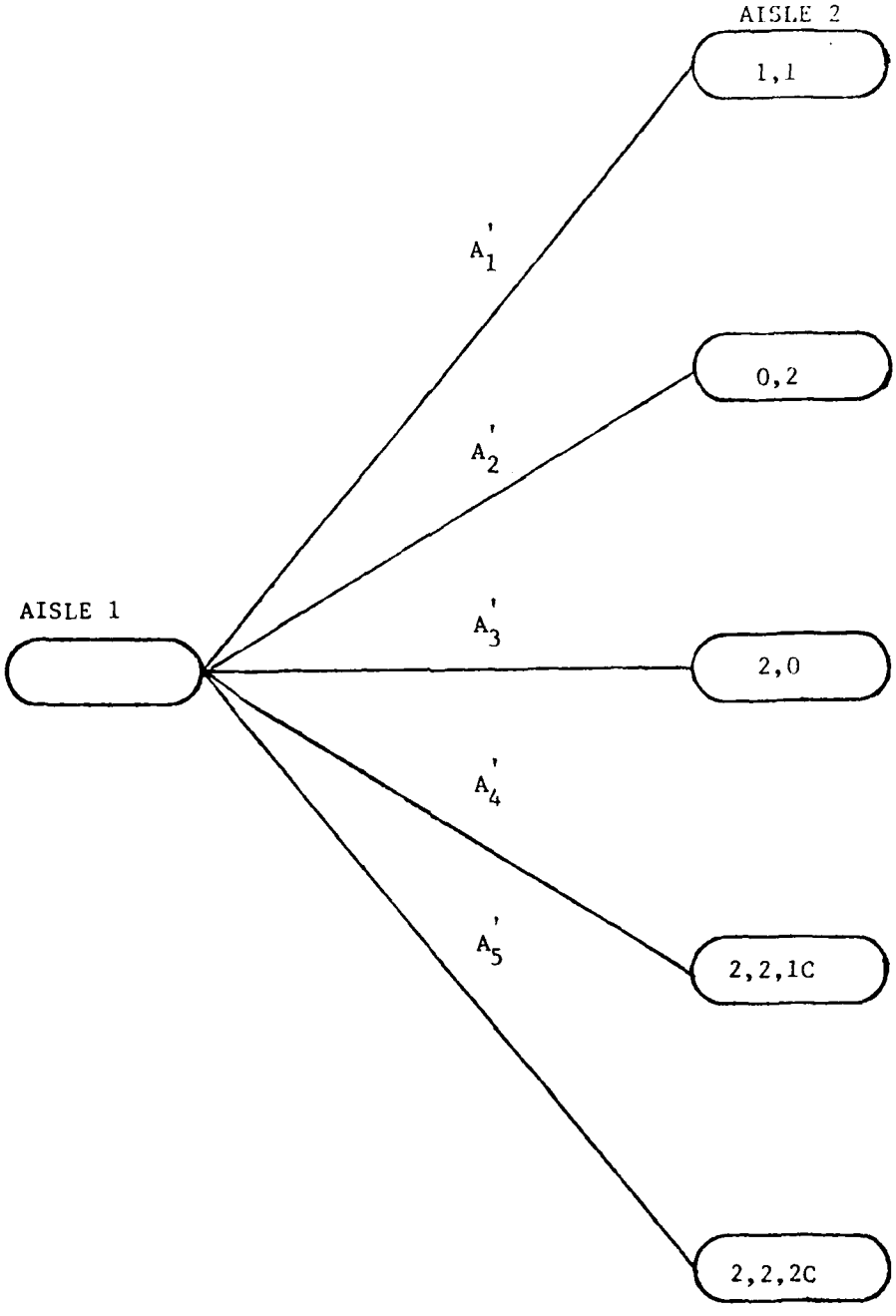


Figure 9: Transitions To An L_2-Partial Tour Subgraph

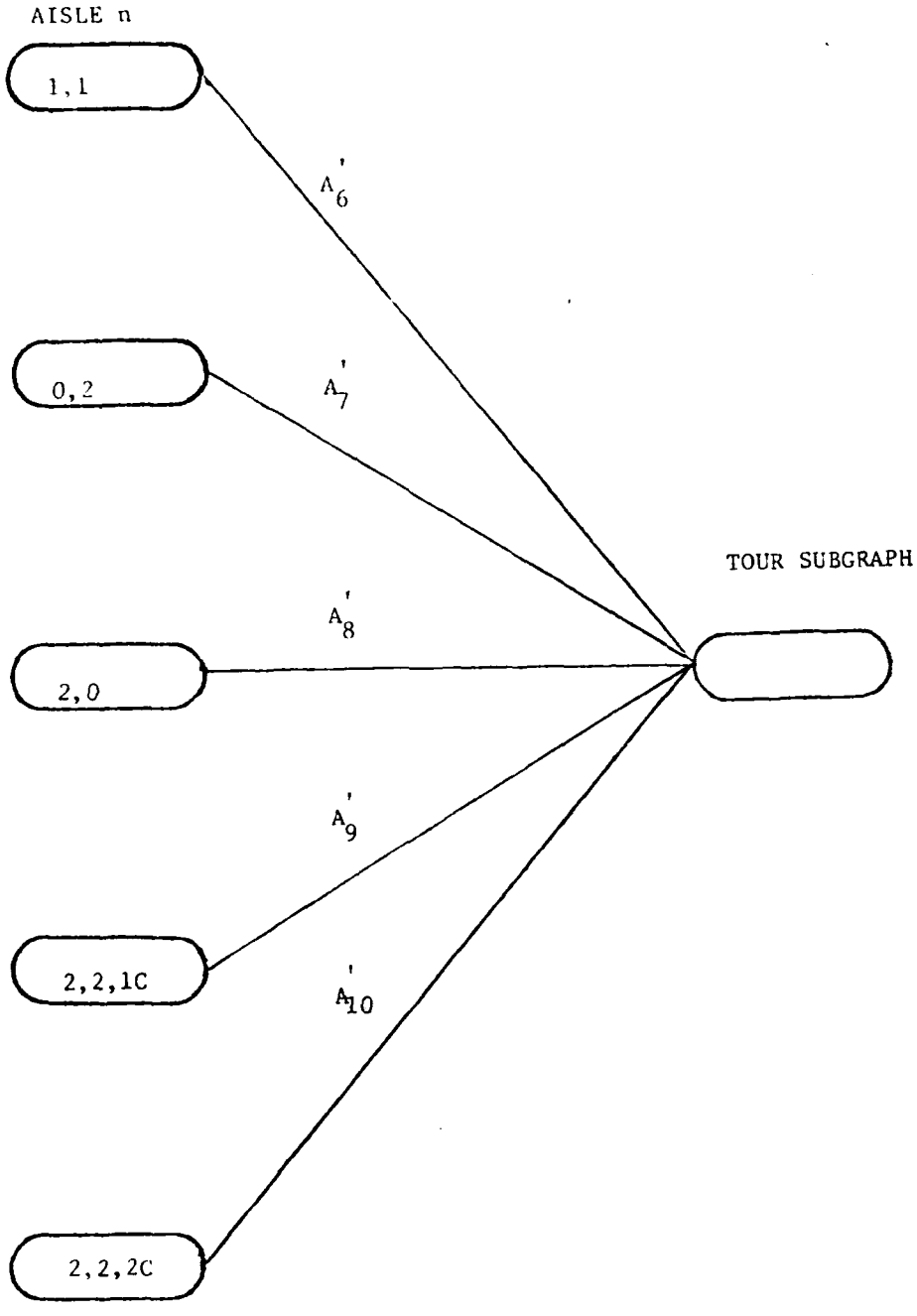


Figure 10: Transitions From A L_n-Partial Tour Subgraph To A Tour Subgraph

Step 5. If there is a pair of parallel arcs in T with one still unused, use it to get to the next vertex. Go to Step 2.

Step 6. Stop. The order-picking tour is complete.

We first note that this procedure constructs a cycle in the tour subgraph T which starts at v_0. Hence, it must also stop at v_0 since each vertex has an even number of edges incident to it. Every edge incident to v_0 must be included in the cycle or else the procedure would not have stopped. The only question left to resolve is whether or not the cycle includes every edge in T. To show this we will use the following results.

Theorem 3: If a_1 and a_2 are a pair of parallel arcs in a minimum length tour subgraph T, there does not exist two arc disjoint cycles in T with a_1 in one and a_2 in the other.

Proof: Suppose that there are two such cycles. If we delete a_1 and a_2 from T, the degree of each vertex in T is still even and T is still connected. This contradicts the assumption that T is a minimum length tour subgraph.

Corollary 3.1: The tour construction procedure cannot include in the cycle only one of the pair of parallel arcs a_1 and a_2.

Proof: Assume that only one of the pair, say a_1 was included in the cycle. We can then construct a second cycle, which includes a_2 by starting with the vertex at one end of a_2 and adding incident edges and vertices not in the first cycle until we reach the vertex on the other end of a_2. From Theorem 3, this contradicts the assumption that T is minimum length tour subgraph.

Theorem 4: The tour construction procedure includes every arc of T in the cycle.

Proof: Assume that some arc of T is not included in the cycle. Since T is connected, there must be an arc, say a, not included in the cycle which has at least one of its vertices, say v' in the cycle. The degree of v' in T is even and the cycle uses an even number of arcs incident to v'. Hence, there are an even number of arcs incident to v' which are not in the cycle. From Corrollary 3.1 any pair of parallel arcs incident to v' are either both in the cycle or neither in the cycle. Therefore, if a' is one of a parallel pair of arcs, neither are in the cycle.

Now consider the last time the cycle visited v'. It could not have left v' via one of a pair of parallel arcs since this would violate Corrollary 3.1 if neither were in the cycle already, and a' would have taken precedence if one of the pair was already in the cycle. Hence, it must have left via a single arc. This implies that a' is also a single arc since it would have taken precedence if it were one of an unused pair.

We see from Figure 6 that a vertex in T has at most two single arcs incident to it. Since v' has incident to it one single arc in the cycle and one single arc not in the cycle and since the number of unused arcs incident to v' is even, there must be a pair of arcs incident to v' with one in the cycle and the other not in the cycles. From Corrollary 3.1 this is a contradiction.

Extensions and Conclusions

The procedure presented here provides a very efficient procedure to solve the order-picking problem for the aisle configuration given in Figure 1. A few variations in this configuration can be handled without much increase in computational effort.

The loading dock can be along any of the aisles without affecting

the model. It can also be at the end of the aisles but not at an intersection. This is most easily handled by inserting an "artifical aisle" with length sufficiently long to keep it from being traversed.

The distance between a_{j-1} and a_j does not have to be the same as the distance between b_{j-1} and b_j. For example, the configuration can be contacted into a circle with radial aisles and still be amenable to the procedure.

If there are items along the ends of aisles as well as within aisles, the basic procedure can still be applied but it is slightly more complex. For the Figure 2 case, the number of arcs in a tour subgraph incident to a_{j-1} and b_{j-1} on the right is the same as the number of arcs incident to a_j and b_j respectively on the left. Hence in setting up the states in Figure 7 we only have states corresponding to degrees to the left of a_{j-1}, b_{j-1}, a_j, and b_j. If there are items to be picked along the ends of aisles, the number of arcs in a tour subgraph incident to a_{j-1} and b_{j-1} on the left is not necessarily the same as the number of arcs incident to a_j and b_j on the right. We can handle this by inserting a set of states in Figure 7 which correspond to the number of arcs incident to a_{j-1} and b_{j-1} on the right. The analysis is then analagous to that done for the previous case. This approximately doubles the computational effort.

The basic ideas can also be extended to the case where cross-overs are allowed within the aisles as well as at the ends of aisles. However this dramatically increases the number of states and cases which must be considered. In the case where cross-overs within aisles were not allowed, we only had to be concerned with combinations of degrees for the two vertices corresponding to the ends of the aisle. If we allow p cross-

overs within an aisle, then we have to consider degree combinations for p + 2 vertices. In addition, we only had to consider one connectivity class for four of the degree combinations and two connectivity classes for the fifth in modelling the problem with no cross-overs within aisles. If we allow cross-overs within aisles the number of connectivity classes also increases. For example, if one cross-over is allowed within each aisle, these are the 13 degree combinations (1,1,2), (1,2,1), (2,1,1), (2,2,0), (2,0,2), (0,2,2), (2,2,2), (0,0,2), (0,2,0), (2,0,0), (1,1,0), (1,0,1), (0,1,1). The first six can have either one or two connected components. The last six have one connected component. The combination (2,2,2) can have either one, two, or three connected components. In addition, to specify the equivalence class if there are two connected components, we need to know which pair of the three intersertion nodes are in the same component. Hence the (2,2,2) case requires five equivalence classes. This is a total of 21 equivalence classes. This can still be handled with a reasonable amount of computational effort. However, the procedure does not seem practical for more than two or three cross-overs with each aisle.

V. SUMMARY

In our three areas of primary focus we feel that we have made a significant fundamental contribution. We have developed and partially tested interactive methodology for addressing the fleet scheduling problem. In addition to greatly improving the fleet scheduling process, insight gained from this effort has provided an exciting new direction for research into other complex scheduling problems. One of these is the scheduling of dry bulk carrying ships for which we have a research effort underway. Another is the scheduling of ship construction into shipyards. This is an area that we are exploring for research.

Interactive methodology for the delivery problem has been developed and tested. This methodology seems clearly superior to previously available methodology for addressing a broad class of routing and delivery problems.

While we started out trying to develop interactive methodology for order-picking problems, we found that we could develop methodology to solve certain classes of these problems optimally without human interaction. Under the current ONR contract, we are exploring other realistic order-picking problems to determine if efficient methods can also be developed for them.

In addition a number of new research areas were indentified and are currently being examined within the Production and Distribution Research Center at Georgia Tech. The Center was established under funding by the Office of Naval Research. These areas include interactive facility design, storage and retrieval system design, and packing and packaging. The basic methodology being developed for these problems can be considered as extensions of interactive concepts and models generated under this contract.

In addition to an ongoing interaction with the SURFPAC scheduler at Norfolk, we have been involved in a number of other briefings and discussions with Navy personnel related to problems of interest to the Navy, particularly

in the area of material handling.

On July 14, 1979 a briefing on material handling systems design was provided at NSC-Oakland to approximately 36 U. S. Navy personnel from NSC-Oakland, NSC-Puget Sound, NSC-San Diego, NAS-Alameda, NARF-Alameda, Mare Island Shipyard, and the Navy School of Transportation. The majority of those attending were senior officers, additionally, several supply officers were present. The briefing was provided with the objective of preparing the executive managers for the design, development, and management of advanced material handling systems required to meet the changing business requirements of the U. S. Navy.

On July 15, 1979 a meeting was held with NSC-Oakland personnel to review ongoing projects involving the analysis and design of automated material handling systems. The projects were critiqued and recommendations were made for future analyses and issues to be addressed.

On September 24-25, 1979 briefings were provided in Washington, D. C. to NAVSUPSYSCOMHG personnel. The first briefing was for the Chief of the Supply Corp and Commander of the Supply Systems Command and his Executive Board, consisting of Admirals and Captains. Subsequently, an expanded briefing was given to NAVSUPSYSCOM Functional managers and others involved in policy-making. Senior level officers (Captains and Commanders) and professional level civilians attended the briefing. Empahsis was placed on strategic planning, analysis, design, and evaluation.

Researchers who have worked on various projects related to research under this contract include Dr. H. Donald Ratliff, Dr. John A. White, Dr. John J. Jarvis, Dr. Arnie Rosenthal and graduate students Bruce Brownlee, Brian Thorn, Mark Goetschalchx, Frank Cullen, and Richard Sharp.


DATE
FILMED
0-8